CHALMERS
UNIVERSITY OF TECHNOLOGY

Effects of measurements on correlations of software code metrics

Empirical Software Engineering (2019) 24:2764–2818
https://doi.org/10.1007/s10664-019-09714-9

Effects of measurements on correlations of software code metrics

Md Abdullah Al Mamun[1] · **Christian Berger**[1] · **Jörgen Hansson**[2]

Published online: 16 May 2019

Abstract

Context Software metrics play a significant role in many areas in the life-cycle of software including forecasting defects and foretelling stories regarding maintenance, cost, etc. through predictive analysis. Many studies have found code metrics correlated to each other at such a high level that such correlated code metrics are considered redundant, which implies it is enough to keep track of a single metric from a list of highly correlated metrics.

Objective Software is developed incrementally over a period. Traditionally, code metrics are measured cumulatively as cumulative sum or running sum. When a code metric is measured based on the values from individual revisions or commits without consolidating values from past revisions, indicating the natural development of software, this study identifies such a type of measure as *organic*. *Density* and *average* are two other ways of measuring metrics. This empirical study focuses on whether measurement types influence correlations of code metrics.

Method To investigate the objective, this empirical study has collected 24 code metrics classified into four categories, according to the measurement types of the metrics, from 11,874 software revisions (i.e., commits) of 21 open source projects from eight well-known organizations. Kendall's τ-B is used for computing correlations. To determine whether there is a significant difference between cumulative and organic metrics, Mann-Whitney U test, Wilcoxon signed rank test, and paired-samples sign test are performed.

Results The cumulative metrics are found to be highly correlated to each other with an average coefficient of 0.79. For corresponding organic metrics, it is 0.49. When individual correlation coefficients between these two measure types are compared, correlations between organic metrics are found to be significantly lower (with $p < 0.01$) than cumulative metrics. Our results indicate that the cumulative nature of metrics makes them highly correlated, implying cumulative measurement is a major source of collinearity between cumulative metrics. Another interesting observation is that correlations between metrics from different categories are weak.

Communicated by: Martin Shepperd

✉ Md Abdullah Al Mamun
abdullah.mamun@chalmers.se

Extended author information available on the last page of the article.

Conclusions Results of this study reveal that measurement types may have a significant impact on the correlations of code metrics and that transforming metrics into a different type can give us metrics with low collinearity. These findings provide us a simple understanding how feature transformation to a different measurement type can produce new non-collinear input features for predictive models.

Keywords Software code metrics · Measurement effects on correlations · Collinearity · Software engineering · Cumulative measurement

1 Introduction

The exponential growth of software size (Deshpande and Riehle 2008) is bringing in many challenges related to maintainability, release planning, and other software qualities. Thus, a natural demand to predict external product quality factors to foresee the future state of software has been observed. Maintainability is related to the size, complexity and documentation of software (Coleman et al. 1994). Size and complexity metrics are common among other metrics to predict software maintainability (Riaz et al. 2009). Growing software size and complexity have made it increasingly difficult to select features to be implemented in the next product release and have challenged existing assumptions and approaches for release planning (Jantunen et al. 2011).

Validating software metrics has gained importance as predicting external software qualities are becoming more demanding day by day to be able to manage future revisions of software. Researchers have proposed many validation criteria for software metrics over the last 40 years, e.g., a list of 47 criteria is reported in a systematic literature study by Meneely et al. (2013) where one of them is *non-collinearity*. Collinearity (also known as multicollinearity) exists between two independent features if they are linearly related. Since prediction models are often multivariate, i.e., use more than one independent feature or metric, it is important that there is no significant collinearity among the independent features. Collinearity results in two major problems (Meloun et al. 2002). First, it makes a model less useful as individual effects of the independent features on a dependent feature can no longer be isolated. Second, extrapolation is most likely be highly erroneous. Thus, El Emam and Schneidewind (2000) and Dormann et al. (2013) suggested diagnosing collinearity among the independent features for a proper interpretation of regression models.

Many studies have explored correlations between various software metrics such as McCabe's cyclomatic complexity (Landman et al. 2016; Henry and Selig 1990; Henry et al. 1981; Tashtoush et al. 2014; Jay et al. 2009; Meulen and Revilla 2007), lines of code (Landman et al. 2016; Henry and Selig 1990; Tashtoush et al. 2014; Jay et al. 2009; Meulen and Revilla 2007), Halstead's metrics (Henry and Selig 1990; Henry et al. 1981; Tashtoush et al. 2014; Meulen and Revilla 2007) Kafura's information flow (Henry et al. 1981), Number of comments, Meulen and Revilla (2007), etc. Most of these studies have observed that code metrics are highly correlated. However, they do not address whether measurement types of metrics affect their correlations which is the primary difference between these studies and our study. Rather than taking the usual way of checking correlations of code metrics, we focus on finding the reason whether the construction of code metrics (meaning how they are measured) have an influence on their correlations. Such an investigation is fundamental toward understanding collinearity of code metrics. A description of the measurement types used in this study are given below.

- **Cumulative**: This indicates the traditional or the most common way how software code metrics are measured by cumulative sum or running sum. Here, by a revision, we indicate a commit, which is a single entry of source code in a repository. For example, if the number of total Lines Of Code (LOC) written for a project's first three revisions are 50, 30, and 30 consecutively, the corresponding cumulative measures of *ncloc* for the revisions would be 50, 80, and 110.
- **Density**: This measure tells us how representative a measure is within a per unit of artifact with a standard portion. Generally, the unit of density is a ratio. Within the context of code, we consider 100 LOC as a unit, the measurement unit becomes a percentage. Under this consideration, such a metric can take a value from 0 to 100. For example, the metric *comment_lines_density* measures lines of comments per 100 lines of code.
- **Average**: This is the mean value of a measure with respect to artifacts related to a specific type. An example of such a metric is *file_complexity* which measures the mean complexity per file.
- **Organic**: A metric that measures artifact from a single revision or two consecutive revisions without being influenced by any other revisions in the repository is organic. We have introduced the term organic as this measure has no effect from the entire list of unbounded preceding revisions like the cumulative measure. An organic metric can measure purely from a single revision, e.g., *new_lines* measures the lines of code (that are not comments) specific to a single revision. It can be zero in case no new code is added to a revision however, it cannot be negative (like a code churn measure). An organic metric can also measure a single revision relative to its one preceding revision. Since in this case it reflects a change or delta compared to the preceding revision, it can be positive, negative and zero.

The core idea of this study was developed while following a previous study (Mamun et al. 2017) where we focused on the domain-level correlation of metrics from four domains that are size, complexity, documentation, and duplications. In the follow-up study, we explored correlations at the metric-level and observed that the organic metrics consistently have lower correlations. Based on this observation from the follow-up study, we initiated and designed this study by grouping the code metrics based on how they are measured.

Due to the problems of collinearity when building predictive models, many studies have investigated how different metrics are correlated with each other. However, to our knowledge, no study has investigated the impact of measurement types on the correlations of software code metrics. This knowledge is fundamental to understand the metrics better. With a goal to understand the relationship between measurement types and correlations of software code metrics, this study has the following research question.

- **RQ**: How measurement types affect correlations of code metrics?

This study has selected 21 open source projects from eight organizations and analyzed the source code of a total 11,874 revisions from all projects to extract code metrics. We have mined 24 software metrics classified into four categories: cumulative, density, average, and organic.

The complete revision histories of the selected projects have been analyzed using a static analysis tool to generate code metrics. The code metrics are then mined from the database for analysis. Before performing data analysis, data is explored using various visual and theoretical statistical tools. Based on the nature of data, we selected Kendall's τ-B (a non-parametric method for correlation), for all selected projects. Motivations for selecting Kendall's τ-B is discussed in Section 4. Correlation coefficients are divided into different sets based on their level of strength and level of significance. Based on the results up to this

point, we transformed all cumulative metrics into organic metrics and ran statistical tests to determine whether there is a significant difference in correlation between these two sets.

Results of this study indicate how correlations of code metrics are influenced by their measurement types, i.e., the way they are measured. We can see whether there is a difference between intra-category correlations of metrics from the same category and inter-category correlations of metrics from different categories. Based on the data analysis, we will also report whether there is a significant difference between intra-category correlations of cumulative metrics and intra-category correlations of organic metrics. These understandings are fundamental because they can reveal whether high collinearity between code metrics are due to their measurement types. Such knowledge can be helpful in making an informed decision while selecting code metrics as features for predictive models.

In the following sections of this paper, we first discuss the methodology including design of this study, data collection procedures, nature of the collected data and data processing. Based on the nature of data observed in Section 3.3, Section 4 (data analysis method), presents a comparative discussion of applicable correlation methods and pros and cons of different measures to aggregate results from the data. Section 5 shows results and implications. Based on some results, this study performed an additional test. Retaining the actual work flow of this study, we have put the design and execution of this test in Section 5.4. This section also includes discussion, limitations and validity threats to this study. Finally, Section 6 summarizes the conclusions of this study.

2 Related Work

Software code metrics are generally known to be highly correlated as many studies have reported high correlation among various code metrics. A recent systematic literature review from 2016 (Landman et al. 2016) presents a summary of 33 articles reporting correlations between McCabe's cyclomatic complexity (McCabe 1976) and LOC (lines of code). Henry and Selig (1990) reported correlations of five code metrics (LOC, three Halstead's software-science metrics (N, V, and E), and McCabe's cyclomatic complexity). They worked with code written in Pascal language and observed three correlations significantly higher that are (the values in parenthesis indicate the correlation coefficients): Halstead N - Halstead V (0.989), LOC - Halstead N (0.893), and LOC - Halstead V (0.885). Henry et al. (1981) compared three complexity metrics: McCabe's cyclomatic complexity, Halstead's effort, and Kafura's information flow. Taking the UNIX operating system as a subject, they found McCabe's cyclomatic complexity and Halstead's effort highly correlated while Kafura's information flow is found to be independent. On NASA's open dataset, Tashtoush et al. (2014) studied cyclomatic complexity, Halstead complexity, and LOC metrics. They found a strong correlation between cyclomatic complexity and Halstead's complexity similar to the study by Henry et al. (1981). LOC is observed to be highly correlated with both of these complexity metrics. Jay et al. (2009), in a comprehensive study, also explored the relationship between McCabe's cyclomatic complexity and LOC. They worked with 1.2 million C, C++ and Java source files randomly selected from SourceForge code repository. They reported that cyclomatic complexity and LOC practically have a perfect linear relationship irrespective of programming languages, programmers, code paradigms, and software processes. Toward comparing four internal code metrics (McCabe's cyclomatic complexity, Halstead volume, LOC, and number of comments), Meulen and Revilla (2007) used 59 specifications each containing between 111 and 11,495 small (up to 40KB file size) C/C++ programs. They observed strong correlations between LOC, Halstead volume, and

cyclomatic complexity. A recent study by Landman et al. (2016) on an extensive Java and C corpora (17.6 million Java methods and 6.3 million C functions) finds no strong linear correlation between cyclomatic complexity and LOC to be considered as redundant. This finding contradicts many earlier studies including (Henry et al. 1981; Tashtoush et al. 2014; Saini et al. 2015; Jay et al. 2009; Meulen and Revilla 2007).

The studies discussed here, mostly cover McCabe's cyclomatic complexity, Halstead's metrics, and LOC investigating correlations between them and showing different results. However, they do not address whether measurement types of the studied metrics affect the strength of correlations which is the primary difference between these studies and our study. Rather than taking the usual way of checking correlations of code metrics, we focus on finding the reason whether the construction of code metrics (meaning how they are measured) have an influence on their correlations. Such an investigation is fundamental toward understanding collinearity of code metrics.

Zhou et al. (2009) have reported that size metrics have confounding effects on the associations between object-oriented metrics and change-proneness. On a revisited study, Gil and Lalouche (2017) reported similar results about the confounding of the size metric. Zhou et al. (2009) have elaborately explained the confounding effect and models to identify them in areas like health sciences and epidemiological research. Gil and Lalouche (2017) used normalization as a way to remove the confounding effect. While they mentioned having lower correlation coefficient for normalized metrics, they have not explicitly reported the overall difference between correlations coming from the intra-cumulative and the intra-*normalized* measures. They also did not report whether there exists a significant statistical difference between the two. But it is understandable as their primary focus is on the validity of metrics. Our focus, in contrast is solely toward understanding the effects of measurements on the correlations of code metrics. We want to understand how much of the collinearity come from the types of measures and how much of it exists naturally.

There have been studies toward understanding the distributions of software metrics. For example, Wheeldon and Counsell (2003), Concas et al. (2007), and Louridas et al. (2008) have investigated whether power law distributions are present among software metrics. They have reported that various software metrics follow different distributions with long fat tails. Louridas et al. (2008) have also reported correlations among eight software metrics including LOC and number of methods. They reported a high correlation between LOC, number of methods (NOM), and out degree of classes. Baxter et al. (2006) reported a similar study, however, unlike Wheeldon and Counsell (2003), have observed some metrics that do not follow the power laws. They opined, their use of a more extensive corpus compared to Wheeldon and Counsell (2003) is the reason for the difference. In addition to looking at the distributions of metrics, Ferreira et al. (2012) have attempted to establish thresholds or reference values for six object-oriented metrics. We have also looked at the statistical properties of the studied metrics including their distributions. However, we have done this as part of out methodology to find appropriate statistical methods, and this is not the main focus of this study.

Chidamber et al. (1998) have investigated six Chidamber and Kamerer (CK) metrics and reported high collinearity among three of them which are coupling between objects (CBO), response for a class (RFC), and NOM. Succi et al. (2005) have studied to what extent collinearity is present in CK metrics. They suggested to completely avoid RFC metric as an input feature for predictive models due to its high collinearity with other CK metrics. Given the problems of collinearity, Shihab et al. (2010) have proposed an approach to select metrics that are less collinear from a set of metrics. These studies have mentioned collinearity as a problem and reported collinearity among software metrics or proposed method to select

metrics with low collinearity. However, they have not investigated from the perspective of measurement types influencing collinearity.

3 Methodology

We have designed this empirical study following the guideline of Runeson and Höst (2008) on designing, planning, and conducting case studies. This study is explorative with the intent to find insights about relations between code metrics with different measurement types. We have designed the study to minimize bias and validity threats and maximize the reliability of the results, which involves project selection, data extraction, data cleaning, exploring nature of the data, select appropriate statistical analysis methods based on the nature of data, and being conservative when selecting and instrumenting statistical analysis.

Data sources for this research are open source software projects, more specifically, open source Java projects on GitHub. Java is among the top three most frequently used project languages on GitHub. Since extracted data is quantitative, analysis methods used in this study are quantitative. We have followed a third-degree data collection method described by Lethbridge et al. (2005). First, the case and the context of the study are defined, followed by data sources and criteria for data collection. Assumptions for statistical methods are thoroughly checked, which involves exploration of the nature of data and cleaning of data as necessary. Regardless of the measurement types, extracted data is non-normal to the extent that meaningful transformation is not possible. Thus, we have used non-parametric statistical methods for analyzing data in this study.

3.1 Project Selection

GitHub's search functionality was used to find candidate projects. However, due to limited capabilities of GitHub search functionality, it was not possible to perform a compound query that would fulfill all our criteria. Project selection was not randomized as we wanted to assure that selected projects have specific criteria (e.g., minimum LOC, minimum commits, etc.) and come from well-known development organizations that would not raise obvious validity questions, e.g., "project is unrepresentative because it is a classroom project by a novice programmer." Thus, finding projects from reputed organizations was exploratory. We started by screening projects from the 14 organizations listed in GitHub's open source organizations showcase[1]. We then explored whether other well-known organizations to our knowledge are also hosting their projects on GitHub but are not in the showcase, e.g., Apache. For each organization, we made queries to find Java projects. As we want to minimize blocking effects coming from various languages, we decided to stick with a single programming language. We selected Java as it is a top-ranked programming language on GitHub.

Crawford et al. (2002) presented various methods for classifying software projects. We take a more straightforward approach to make sure that our selected projects are representative regarding size. A study[2] on the dataset of International Software Benchmarking Standards Group (ISBSG) classified software projects based on "Rule's Relative Size Scale". Measurements of this study are based on IFPUG MkII and COSMIC which are also translated into equivalent LOC. The combined distribution of all projects shows that more than 93% of the projects are between S (small) and L (large) size where S is estimated as

[1] https://github.com/showcases/open-source-organizations

[2] https://www.totalmetrics.com/function-points-downloads/Function-Point-Scale-Project-Size.pdf

Table 1 Overview of the selected projects

Organization	Project name	Contributors (person)	Analyzed Revisions (commits)	Total Revisions (commits)	Java Code (LOC)	Total Code (LOC)	Latest Commit (SHA)	Project Duration (months)
Microsoft	oauth2-useragent	6	171	336	2,976	4,059	d5ddee2	12
	Git-Credential-Manager-for-Mac-and-Linux	5	141	679	4,986	6,169	5fcb321	13
	malmo	12	295	814	14,221	33,212	8a1bc7d	5
	thrifty	5	242	299	43,948	44,324	7f8c12a	12
	Vso-intellij	22	305	1,191	64,457	65,502	682cb12	12
Twitter	ambrose	18	167	640	4,879	10,138	da7bcb9	48
	cloudhopper-smpp	15	94	150	12,342	12,452	193d1c4	57
	elephant-bird	52	449	1,361	22,675	26,087	87efd8c	76
Netflix	Fenzo	10	98	192	11,174	12,360	4b446e3	20
	ribbon	30	223	785	22,299	22,634	4522f71	46
	astyanax	51	549	991	55,167	55,680	4324ba7	55

Table 1 (continued)

Organization	Project name	Contributors (person)	Analyzed Revisions (commits)	Total Revisions (commits)	Java Code (LOC)	Total Code (LOC)	Latest Commit (SHA)	Project Duration (months)
Square	dagger	36	306	699	8,894	10,927	9888337	46
	retrofit	103	776	1,378	12,908	14,329	32ace08	72
	picasso	73	518	931	9,960	11,094	a1ba906	42
Esri	solutions-geoevent-java	7	218	535	34,981	60,752	f4f872e	38
	geometry-api-java	14	100	174	76,114	76,391	3858559	43
Shopify	nokogiri	105	1,788	3,513	26,398	61,129	e2821be	75
SAP	Cloud-sfsf-benefits-ext	9	52	165	3,029	5,638	984de61	24
Apache	kafka	236	2,302	2,863	88,920	155,260	8f2e0a5	64
	zookeeper	9	1,474	1,474	72,894	137,466	f6349d1	109
	zeppelin	158	1,606	2,707	64,878	100,551	ba2b90c	39
Total		976	11,874	21,877	658,100	926,154		907
Mean		46	565	1,042	31,338	44,103		43

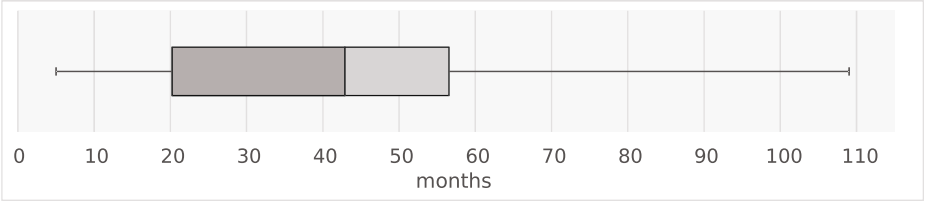


Fig. 1 Whiskers box plot showing durations of the selected projects

5300 and L as 150,000 LOC. We roughly followed this finding and selected projects in a way that project sizes are about uniformly distributed within about the range of S and L. We have projects ranging from 4059 LOC to 155,260 LOC indicating the code size of the latest revision of projects. Sizes of the projects are determined with *cloc* tool[3] using a bash script to extract total LOC and Java LOC. We selected 21 GitHub projects from eight software organizations where Java is tagged as the project language.

An overview of the selected projects is given in Table 1. In this table, 'analyzed revisions' indicates all the commits from which we collected data and which are available exclusively in the master branch of the Git repositories. 'Total revisions' indicates all commits available in the Git repository including the branches. Even though the selected projects are classified as Java projects, they have source code from other languages too. Thus, 'total code' indicates the amount of all lines of code and 'Java code' indicates only the lines of Java source code. The table field 'latest commit' points to the Head of a Git repository at the time we downloaded it. The time durations of the projects are presented in the whiskers box plot in Fig. 1 showing duration of projects from five months to 109 months with a median of 43 months. About 33% of the projects are within the 4^{th}-quartile ranging from 57 to 109 months.

3.2 Data Collection and Metrics Selection

We used SonarQube[4] to analyze revisions of the selected projects. Kazman et al. (2015) mentioned SonarQube as the de-facto source code analysis tool for automatically calculating technical debt. It has gained popularity in recent years, and Janes et al. (2017) mentioned SonarQube as the de-facto industrial tool for Software Quality Assurance. This tool is based on SQALE methodology (Letouzey and Ilkiewicz 2012). We used SonarQube version 6.1 and Sonar-Scanner version 2.8.

We run SonarQube on each revision available in the master branch of a project. Since we ignore sub-branches, the number of analyzed revisions is less than the number of total revisions as reported in Table 1. Sub-branches are eventually merged with the master branch which means, we do not lose anything except the granularity of data.

Analyzing 11,874 software revisions needs be automated. Python scripts are used to automate the process of traversing commits or revisions on the master branch of a project's Git repository and run SonarQube tool on commits. SonarQube provides web-services covering a range of functionalities including mining analysis results and software metrics. We observed some of the metrics such as *new_lines* are seen on SonarQube's web-interface, but

[3] https://github.com/AlDanial/cloc
[4] https://www.sonarqube.org/

Table 2 Classification of the selected code metrics according to "How They Measure"

Measurement Category	Metric name	Description	Value Type
Cumulative	ncloc	Number of physical lines of code that are not comments (line only containing space, tab, and carriage return are ignored)	Integer
	classes	Number of classes (including nested classes, interfaces, enums, and annotations)	Integer
	files	Number of files	Integer
	directories	Number of directories	Integer
	functions	Number of methods	Integer
	statements	Number of statements according to Java language specifications	Integer
	public_api	Number of public Classes + number of public Functions + number of public Properties	Integer
	comment_lines	Number of lines containing either comment or commented-out code (Empty comment lines and comment lines containing only special characters are igonored)	Integer
	public_undocumented_api	Public API without comments header. Public undocumented classes, functions and variables	Integer
	complexity	Cyclomatic complexity (else, default, and finally keywords are ignored)	Integer
	complexity_in_classes	Cyclomatic complexity in classes	Integer
	complexity_in_functions	Cyclomatic complexity in functions	Integer

Table 2 (continued)

Measurement Category	Metric name	Description	Value Type
	duplicated_lines	Number of duplicated lines	Integer
	duplicated_blocks	Number of duplicated blocks. To count a block, at least 10 successive duplicated statements are needed. Indentation & string literals are ignored	Integer
	duplicated_files	Number of duplicated files	Integer
Density	comment_lines_density	*comment_lines_density* = *comment_lines* / (*ncloc* + *comment_lines*) * 100	Percent
	public_documented _api_density	*public_documented_api_density* = (*public_api* - *public_undocumented_api*) / *public_api* * 100	Percent
	duplicated_lines_density	*duplicated_lines_density* = *duplicated_lines* / *ncloc* * 100	Percent
Average	file_complexity	Average complexity by file. *file_complexity* = *complexity* / *files*	Float
	class_complexity	Average complexity by class. *class_complexity* = *complexity_in_classes* / *classes*	Float
	function_complexity	Avgerage complexity by function. *function_complexity* = *complexity_in_functions* / *functions*	Float
Organic	new_lines	Number of new physical lines of code that are not comment	Integer
	new_duplicated_lines	Number of new physical lines of code that are duplicated	Integer
	new_duplicated_blocks	Number of new blocks of code that are duplicated	Integer

Table 3 Metric data representing five revisions of a project

	Cumulative															Density			Average			Organic		
Revision No.	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
88	9573	721	5082	2354	2354	2349	102	70	626	504	1080	6	383	29	9	10.1	19.5	4.0	33.6	23.1	3.26	2	0	0
89	9590	723	5095	2360	2360	2355	102	70	627	505	1081	6	383	29	9	10.1	19.5	4.0	33.7	23.2	3.26	25	0	0
90	9642	725	5124	2375	2375	2370	103	71	629	506	1088	6	383	29	9	10.1	19.6	4.0	33.5	23.1	3.27	70	0	0
91	11297	801	6035	2647	2647	2641	120	81	679	552	1137	7	435	30	10	9.1	18.7	3.9	32.7	22.1	3.30	1845	52	1
92	11297	801	6035	2647	2647	2641	120	81	679	552	1137	7	435	30	10	9.1	18.7	3.9	32.7	22.1	3.30	0	0	0

they cannot be mined through the web-services. We later found that SonarQube computes some metrics only for the latest software revision and removes them automatically.

Since we did not find any option to stop the auto-deletion, we added triggers and additional tables into SonarQube's SQL database to recover the deleted records. In total, 47 metrics were mined from the database classified into six major domains namely size, documentation, complexity, duplications, issues, and maintainability. The classification is based on what the metrics measure.

In our earlier study (Mamun et al. 2017), we used SonarQube's classification of metrics and explored domain-level and metric-to-domain-level relationships. From the results of the metric-to-domain-level relationships in that study, we had the indication that metrics that measure artifacts based on individual values from each revision (i.e., organic metrics), result in lower overall correlation. Since metrics such as *new_lines* of type organic are inherently different from metrics such as *ncloc* of type cumulative concerning how they measure artifacts, it was understandable. However, as we started the follow-up study exploring the metric-level correlations, we observed that organic metrics have much lower correlations compared to other types of metrics. This observation influenced us to rethink how the metrics should be grouped for comparison. So the criteria to group the metrics changed from earlier "what they measure" to "how they measure" artifacts. We looked at the metrics classified into four domains (i.e., size, complexity, documentation, and duplications) based on "what they measure" by SonarQube. Reviewing them, we identified 24 metrics of four measurement types that are cumulative, density, average, and organic. Table 2 shows the selected metrics classified into these four measurement types along with a short description and value type, taken from the MySQL database of SonarQube 6.1 and the metric definitions page.[5]

Table 3 shows metrics data corresponding to five revisions or commits of a project. In this table, each row represents a software revision. For project *malmo*, we have analyzed 295 revisions; thus, we have 295 data rows from this project with the similar structure as Table 3. Data used for this study is measured at the project-level. For example, for *ncloc*, all lines of Java code in the entire project is counted, for classes, all classes within the scope of a project are counted. In Table 3, the value of *ncloc* for the whole project is 9573 for revision 88. In the next revision (i.e., 89), *ncloc* becomes 9590 indicating an increase of 17 lines of code. However, the corresponding *new_lines* metric for this revision is 25 indicating the actual number of lines of code added to this revision disregarding possible changes or deletions of the code. All the metrics are calculated by SonarQube according to the description in Table 2. Among the four categories, density and average metrics are

[5] https://docs.sonarqube.org/latest/user-guide/metric-definitions/

Table 4 Considered methods for normality test

	Graphical methods	Numerical methods
Descriptive	Histogram, box plot	Skewness, Kurtosis
Theory-driven	Q-Q plot	Shapiro-Wilk, Kolmogorov-Smirnov test (Lillefors test)

derived metrics, meaning, they are measured based on the cumulative metrics. Equations for constructing density and average metrics are given in Table 2.

SQL code to instrument the SonarQube database, Python code to automatically analyze commits of a Git project with SonarQube, MySQL code to retrieve the desired data from the database, and the collected data used for this study are published as public dataset[6].

3.3 Exploring Nature of Data

Among different probability distribution functions, a normal distribution is more anticipated by the researchers due to its relationship with the natural phenomenon "central limit theorem." Statistical methods are based on assumptions. After collecting data, before deciding on the type of statistical methods, researchers need to investigate the nature of the collected data. A crucial part is to check the distribution of data. If the distribution is normal, parametric statistical tests are considered. If data is non-normal and a meaningful transformation is not possible, non-parametric tests are considered.

There is no straightforward way of determining whether a particular data is normally distributed. Sample size plays a significant factor in statistical tests for normality. There are graphical and numerical methods where each of them can be either descriptive or theoretical (Park 2009). Since our selected projects have a varying number of revisions, we have chosen a combination of tests appropriate to our data as shown in Table 4.

Histogram, a frequency distribution, is considered to be a useful graphical test when the number of observations is large. It is particularly helpful because it can capture the shape of the distribution given that bin size is appropriately chosen. If data is far from a normal distribution, a single look at the histogram tells us that the data is non-normal. It also gives a rough understanding of the overall nature of the distribution, such as skewness, kurtosis or the type of distribution such as bi- and multi-modal, etc. Box plots are useful to identify outliers and comparing the distribution of the quartiles. Normal Q-Q plot (quantile-quantile plot) is a graphical representation of the comparison of two distributions. If data is normally distributed, the data points in the normal Q-Q plot approximately follow a straight line. It also helps us to understand the skewness and tails of the distribution. The graphical methods help to understand the overall nature of the data quickly, but it does not provide objective criteria (Park 2009).

There are different numerical methods to evaluate whether data is normally distributed or not. Skewness and kurtosis are commonly used descriptive tools for this purpose. For a perfectly normal distribution, the statistics for these two analyses should be zero. Since this does not happen in practice, we calculate the *z-score* by dividing the statistic by the standard error. However, determining normality from *z-score* is not straightforward either. Kim (2013) discussed how the sample size can affect the *z-score*. Field (2009) and Kim (2013) suggested to consider different criteria for skewness and kurtosis based on the sample size.

[6] https://archive.org/details/EoMoCoSCM

Table 5 Descriptive statistics for four example metrics from the Apache *zookeeper* project

Category	Metrics	N	Minimum	Maximum	Mean		Std. Dev.
					Statistic	Std. Error	
Cumulative	ncloc	1473	10,804	73,009	44,192.4	485.3	18,626.9
Density	comment_lines_density	1473	9.2	13.9	12.5	0.0	1.2
Average	file_complexity	1473	22.0	33.9	25.6	0.1	2.7
Organic	new_lines	1473	0	19,055	117.5	16.3	627.3

For a sample size less than 50, an absolute *z-score* for either of these methods should be 1.95 (corresponding to an α of 0.05); for a sample size less than 200, an absolute *z-score* of 3.29 (corresponding to an α of 0.001). However, for a sample size of 200 or more, it is more meaningful to inspect the graphical methods and look at the skewness and kurtosis statistics instead of evaluating the significance, i.e., *z-score*.

From analytical numerical methods, we consider Shapiro-Wilk and Kolmogorov-Smirnov for normality test. Shapiro-Wilk works better with sample size up to 2000 and Kolmogorov-Smirnov works with large sample sizes (Park 2009). The literature has reported different maximum values for these tests. For example, sample size over 300 might produce an unreliable result for these two tests, observed by Kim (2013), and a range of 30 to 1000 is suggested by Hair (2006). We have sample sizes from 52 to 2302 for our metrics. For large sample size, numerical methods can be unreliable; thus, Field (2009) suggested to use the graphical methods besides the numerical methods to make an informed decision about the nature of the data. We have computed all these tests in the statistical software package SPSS. It can be noted that SPSS calculates $Kurtosis - 3$ for Kurtosis, meaning subtracting three from the Kurtosis value.

Now, we present data of four metrics (*ncloc*, *comment_lines_density*, *new_lines*, and *file_complexity*), each from a different measurement category, from the Apache *zookeeper* project. Even though measures for these metrics vary in each project but we present them here so the readers have a rough idea how sample data from a project might look like. Table 5 shows the descriptive statistics for four metrics from four measurement types. The sample size is the same, i.e., 1473 for all these metrics and minimum, maximum, mean, and standard deviation values come from the whole sample. For example, among 1473 samples of *new_lines*, it has the minimum value 0 (i.e., a revision that adds no new lines of code) and

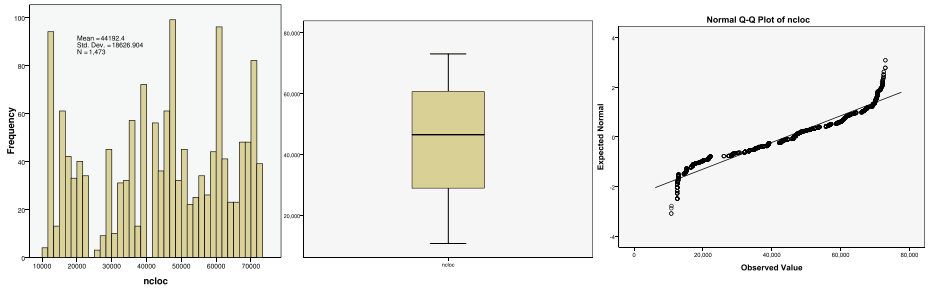


Fig. 2 Histogram, box plot and normal Q-Q plot for *ncloc* metric (cumulative type) from the Apache *zookeeper* project

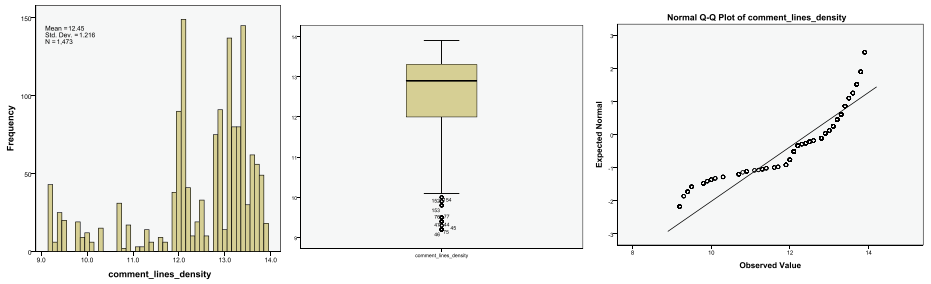


Fig. 3 Histogram, box plot and normal Q-Q plot for *comment_lines_density* metric (density type) from the Apache *zookeeper* project

maximum value 19,055 (i.e., a revision that adds 19,055 new lines of code). Based on these statistics, it is evident that *ncloc* is entirely different from *new_lines*. On the other hand, two metrics from density and average categories are quite similar but different from cumulative and organic metrics. The most notable number in this table is the tremendous 18,629.9 value of standard deviation for *ncloc*. The cumulative way of measurement and the large sample size are the key reasons for such a high standard deviation. Other cumulative metrics in our dataset also have a similar effect of cumulation on them. These descriptive statistics give us a quick overall idea about the nature of data, e.g., distribution, dispersion.

Histogram, box plot, and normal Q-Q plot for each of these four metrics are correspondingly presented in Figs. 2, 3, 4, and 5. The organic metric *new_lines* (see Fig. 2) has a very high peak (in the histogram), a considerable amount of outliers (in box-plot) and a positively-skewed plot (normal Q-Q). The other metrics from the organic category have similar properties. Because of the very high peak in the distribution, the quartiles in the box-plot are not even visible. The *file_complexity* metric of type average (see Fig. 4) has a bimodal distribution in the histogram. The outliers in the box plot and the disconnected observed values in the normal Q-Q plots are due to the bimodal distribution of *file_complexity* metric. It is interesting to see in the Q-Q plot that both distributions for *file_complexity* are skewed in opposite directions. Many metrics in average and density, and even in the cumulative category have bimodal distributions, and some have multimodal distributions. The distributions of the average and density metrics are apparently more normal compared to the distributions of the cumulative metrics. However, all of them are still far away from a normal distribution. If we compare them all, organic metrics are distinctly different compared to the metrics from other categories. When specific types of plots are compared, e.g., all histograms of

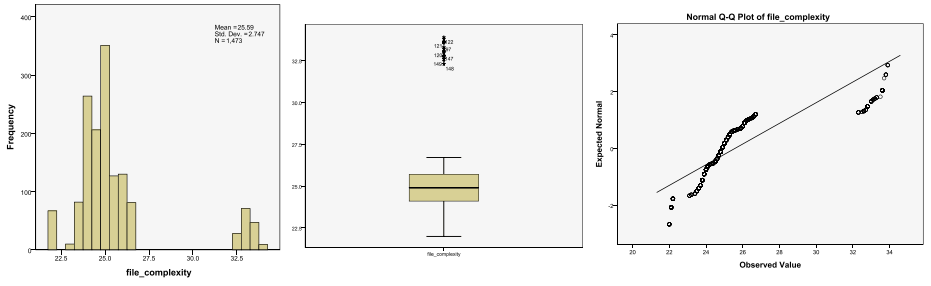


Fig. 4 Histogram, box plot and normal Q-Q plot for *file_complexity* metric (average type) from the Apache *zookeeper* project

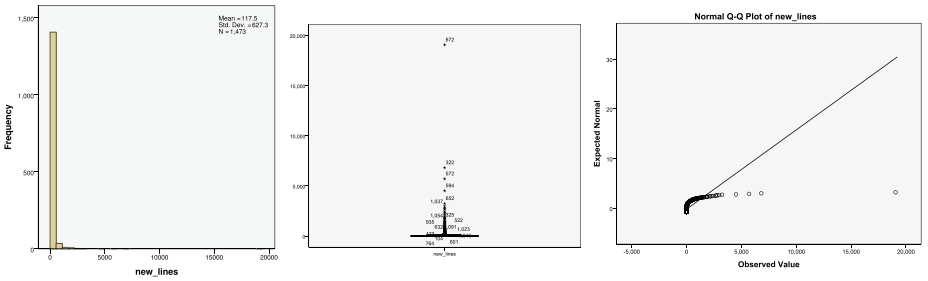


Fig. 5 Histogram, box plot and normal Q-Q plot for *new_lines* metric (organic type) from the Apache *zookeeper* project

metrics from a category, plots of the organic metrics are visually observed to be more consistent to each other compared to the plots of metrics from other categories. On the other hand, when plots are compared across categories, distributions of metrics from density and average categories are observed to be more similar than others.

The depiction of the distributions and their properties by graphical methods are so much deviated from normality that we could have omitted additional tests for them. However, we perform them as part of the design of the study. Since we have a varying number of sample sizes, we got some insights about the tests. However, such observations are beyond the scope and focus of this study, thus, are not reported in this report.

Skewness and Kurtosis results are presented in Table 6. It is interesting that even though, we have a considerable sample size, none of the metrics are indicating normal distribution in the *z-scores* in this table, meaning data is non-normal because absolute z-scores are greater than 3.29, thus, rejecting the null hypothesis. However, compared to these four metrics, the *ncloc* and *file_complexity* from the Apache *kafka* project having the largest sample size of 2302 have *z-scores* within the limit of normality, but we reject it because the graphical tests do not show signs of a normal distribution.

We considered Shapiro-Wilk and Kolmogorov-Smirnov tests reliable up to a sample size of 2000. Results from these tests are depicted in Table 7 showing a very strong indication (i.e., the significance values are less than the considered $\alpha = 0.05$) of non-normal data.

Likewise, these four metrics from the Apache *zookeeper* project, other metrics from these measurements categories or from other projects are also observed to be non-normal. Due to a high degree of non-normality and different distributions among the metrics make it impractical to make transformations on these metrics. Thus, we are left with the option to perform non-parametric statistical analysis methods.

Table 6 Skewness and Kurtosis check for four example metrics from the Apache *zookeeper* project

Metrics	Skewness			Kurtosis		
	Statistic	Std. Error	z-value	Statistic	Std. Error	z-value
ncloc	−0.24	0.06	−3.71	−1.13	0.13	−8.87
comment_lines_density	−1.30	0.06	−20.40	0.91	0.13	7.11
file_complexity	*2.00*	0.06	31.39	*3.15*	0.13	24.68
new_lines	*20.83*	0.06	326.63	*580.39*	0.13	4,554.62

Table 7 Shapiro-Wilk and Kolmogorov-Smirnov tests for four example metrics from the Apache *zookeeper* project

	Kolmogorov-Smirnov*			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
ncloc	.098	1473	.000	.939	1473	.000
comment_lines_density	.181	1473	.000	.842	1473	.000
file_complexity	.258	1473	.000	.693	1473	.000
new_lines	.426	1473	0.000	.150	1473	.000

*. Lilliefors Significance Correction

3.4 Data Processing

Besides the statistical analysis to understand the nature of the extracted data, we performed manual inspections to check the data for possible anomalies especially at the boundaries meaning, at the beginning and the ending of revision data.

We observed some organic metrics for some projects have either NaN (not a number) or unusually high value for the first analyzed revision. Examples of such observations are presented in Tables 8 and 9.

There might be several reasons why some projects start with a higher number of *ncloc* from the beginning as we see in the case in Table 9. It can be due to a project not tracking its code base through a version control from the beginning, or the project start with an existing code base possibly because it is an extension of another project with a separate version control. It is also observed that *new_lines* has a higher value than *ncloc* in the first revision of few projects. We removed data related to the first revisions in such cases. Since the number of removed revisions is very insignificant compared to the total number of revisions, it is less likely that this will have a major impact on this study.

4 Data Analysis Method

Since collected data is not normally distributed, non-parametric statistical methods are appropriate for this research. Spearman's ρ correlation coefficient and Kendall's τ correlation coefficient are two well-known non-parametric measures to assess relationships between ranked data. We carefully investigated the nature of the collected data and properties of these two measures. A measure of Kendall's τ is based on the number of concordant and discordant pairs of the ranked data.

Calculating Spearman's ρ by hand is much simpler than calculating Kendall's τ because it can be done pair-wise without being dependent on the rest of the data while computing Kendall's τ by hand is only feasible for small sample sizes because computing each data pair requires exploring the remaining data. Xu et al. (2013) reported that the time complexity

Table 8 Data snippet from the first revision of project *astyanax* showing null value for *new_lines*

ncloc	New_lines	Classes	Files	directories
8409	NULL	170	157	12

Table 9 Data snippet from the first three revisions of project *ribbon* showing a high value for *new_lines*

ncloc	New_lines	Classes	Files	Directories
6111	8461	80	61	9
6118	42	80	61	9
5981	15	79	60	9

of Spearman's ρ is $O(n * \log(n))$ and for Kendall's τ it is $O(n^2)$. In general, Spearman's ρ results in a higher correlation coefficient compared to Kendall's τ where the latter is generally known to be more robust and has an intuitive interpretation.

Studies from different disciplines have investigated the appropriateness of Spearman's ρ and Kendall's τ concerning various factors. Both measures are invariant concerning increasing monotone transformations (Kendall and Gibbons 1990). Moreover, being non-parametric methods, both measures are robust against impulsive noise (Shevlyakov and Vilchevski 2002; Croux and Dehon 2010). Croux and Dehon (2010) studied the robustness of Spearman's ρ and Kendall's τ through their influence function and gross-error sensitivities. Even though it is commonly known that both of these measures are robust enough to handle outliers, this study found that Kendall's τ is more robust to handle outliers and statistically slightly more efficient than Spearman's ρ. In a more recent study, Xu et al. (2013) investigated the applicability of Spearman's ρ and Kendall's τ based on different requirements. Some of their key findings report Kendall's τ as the desired measure when the sample size is large, and there is impulsive noise in the data. Their results are based on unbiased estimations of Spearman's ρ and Kendall's τ correlation coefficients.

In light of the above discussion, we think, Kendall's τ is more appropriate for software projects. We have observed outliers in many projects, and all projects have some revisions with high data values indicating outliers. This can naturally happen to any software project when existing code-base is added to a new project. Outliers in software revision data are observed and their underlying reasons are discussed in earlier research (Aggarwal 2013; Schroeder et al. 2016). The robust nature of Kendall's τ handles such data points better when compared to Spearman's ρ.

Kendall's τ has three different versions that are τ-A, τ-B, and τ-C. Both τ-A and τ-B are suitable for square-shaped data meaning data with same variables on both rows and columns. τ-C is used for rectangular-shaped data tables with different sized rows and columns. A more important difference between τ-A and τ-B is that τ-B can handle tied ranks while having otherwise the same characteristics of τ-A. Since our collected data has tied ranks, we have used and measured τ-B according to the following equation[7]:

$$\tau_b = \frac{P - Q}{\sqrt{(P + Q + m1_0)(P + Q + m2_0)}}$$

In this equation, *m1*, *m2* are the two metrics for which we are checking correlation. We denote correlation coefficients of Kendall's τ as τ_b. P, Q are numbers of concordant and discordant pairs respectively. $m1_0$ is the number of ties only in *m1*, and $m2_0$ is the number of ties only in *m2*. The possible value of τ_b is $-1 \leqslant \tau_b \leqslant +1$ where -1 indicates perfect negative correlation, zero indicates no correlation, and +1 indicates perfect positive correlations. Since we have 21 projects and 24 metrics in each project, we have calculated 21

[7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html

Table 10 Grouping and labeling τ_b

Statistical significance	Negative τ_b value	Positive τ_b value	Label
$\alpha = 0.05$	$-1.0 \leqslant \tau_b \leqslant -0.9$	$0.9 \leqslant \tau_b \leqslant 1.0$	Very strong τ_b
	$-0.9 < \tau_b \leqslant -0.7$	$0.7 \leqslant \tau_b < 0.9$	Strong τ_b
	$-0.7 < \tau_b \leqslant -0.4$	$0.4 \leqslant \tau_b < 0.7$	Moderate τ_b
	$-0.4 < \tau_b \leqslant -0.0$	$0 \leqslant \tau_b < 0.4$	Weak τ_b

correlation matrices of size 24×24. All these correlation matrices are symmetric meaning they are mirrored along the principal diagonal. τ_b for the diagonal elements are always +1, indicating a perfect positive correlation of a metric with itself. Kendall's τ also computes statistical significance (*p-value*) for correlation coefficient (τ_b).

4.1 Landscape of Correlation Coefficients (τ_b)

Before start aggregating results, we need to understand τ_b and define concrete boundaries for the interpretation of τ_b at different levels. τ_b indicates the strength of a correlation and τ_b has the range $-1 \leqslant \tau_b \leqslant +1$. As the value of τ_b approaches 0, it indicates less correlation between two metrics. As the value of τ_b approaches to the boundaries, i.e., -1 or +1, it indicates a higher correlation between two metrics. Researchers have used different ranges to label the strengths of correlation coefficients (Taylor 1990). This paper has labeled τ_b according to Table 10.

However, it is not enough just to look at the τ_b values, unless we look at their statistical significance, which can be found from the *p-value*. If the *p-value* is greater than a chosen significance level, we cannot reject the null hypothesis, meaning we do not have enough evidence to differentiate that a corresponding τ_b is any different from $\tau_b = 0$. For example, for a *p-value* value of 0.05, there is a possibility that 5% of the τ_b values are indicating correlation by chance even though there is no real correlation between the underlying metrics. This study considers $\alpha = 0.05$ for any τ_b to be statistically significance.

Now we like to focus on the landscape of τ_b depicted in Fig. 6. The outer circle is the set of all τ_b, which is represented by $S\tau_{b_{all}}$ containing τ_b from correlation matrices resulted from all projects. Similarly, the set of all significant τ_b is represented by $S\tau_{b_{sig}}$ containing all τ_b, for which the corresponding *p-value* has satisfied the specified significance level of 0.05. Set of very strong τ_b is represented by $S\tau_{b_{vs}}$, set of strong τ_b by $S\tau_{b_s}$, set of moderate τ_b by $S\tau_{b_m}$, and set of weak τ_b by $S\tau_{b_w}$. These sets are calculated according to τ_b values based on the levels defined in Fig. 10. Now we can also define the set of all non-significant τ_b as $S\tau_{b_{nsig}}$ where $S\tau_{b_{nsig}} = S\tau_{b_{all}} \setminus S\tau_{b_{sig}}$.

4.2 Aggregating Correlation Coefficients (τ_b)

Now that we have 21 metrics of size 24x24, we want to aggregate meaningful data from them. Let $M = \{m_1, m_2, m_3 ... m_n\}$, where n is the number of total metrics be the set of all metrics and $P = \{p_1, p_2, p_3 ... p_q\}$, where q is the number of total projects be the set of all projects.

Now we discuss several ways to aggregate τ_b from all the projects.

– Aggregating correlation coefficients (τ_b) based on the set of all correlation coefficients ($S\tau_{b_{all}}$)

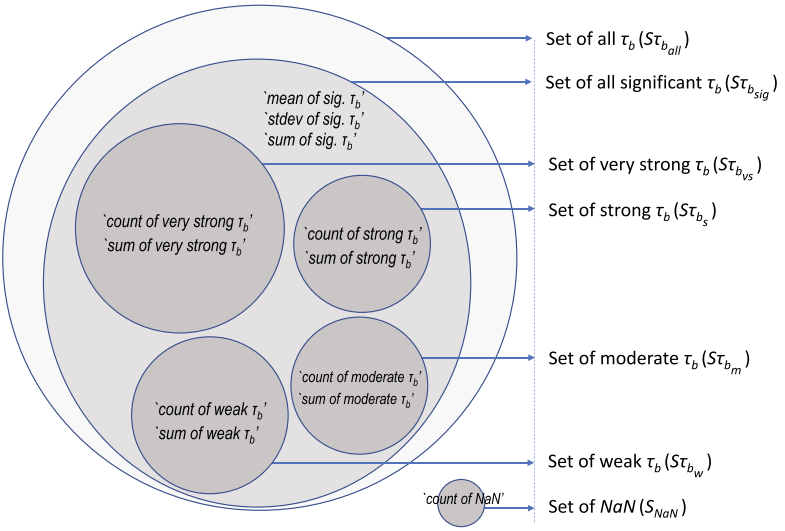


Fig. 6 Sets of correlation coefficients (τ_b). Text inside the circles indicates measures computed from τ_b

The simplest way of aggregating τ_b is by summing up all τ_b values for each possible pair of metrics within M for each project within P. This results in a correlation matrix of size 24×24, i.e., the same size of a correlation matrix from a project. The advantage of this method is that we can compute a single aggregated matrix where each cell is the sum of τ_b values from the corresponding cells from correlation matrices generated from the projects. Such an aggregated matrix can also be transformed into a weighted average matrix by dividing each cell (containing the sum of all τ_b for a particular pair of metrics) by q.

However, there is a fundamental problem with this method. This method combines all τ_b values irrespective of their corresponding *p-value*. If the *p-value* is greater than α then we cannot reject the null hypothesis. This means, we only consider a τ_b valid when its corresponding *p-value* is less than or equal to α. If this is not checked, there are two implications. First, a result will be wrong due to the inclusion of the non-significant τ_b or inclusion of τ_b from the set $S\tau_{b_{nsig}}$; second, we will not have any idea how big the set $S\tau_{b_{nsig}}$ is compared to $S\tau_{b_{sig}}$.

Aggregating results based on $S\tau_{b_{all}}$ is not problematic in the case when $S\tau_{b_{all}} = S\tau_{b_{sig}}$, meaning there is no τ_b with a non-significant *p-value*, which is an assumption that should be checked for validity in case it is assumed. For example, the recent research on the correlation of code metrics by Gil and Lalouche (2017) has not reported anything about considering α, i.e., the significance level for *p-value*, thus, nothing about the existence of τ_b with a non-significant *p-value*. In this case, the readers cannot know the size of $S\tau_{b_{nsig}}$. If an assumption regarding the non-existence of non-significant *p-value* is considered, then that should be documented and validated. In this research, to aggregate results, we have avoided any value from the set $S\tau_{b_{nsig}}$. To calculate the sample mean, τ_b values from $S\tau_{b_{nsig}}$ are considered as zero.

– Aggregating correlation coefficients (τ_b) based on the set of all significant correlation coefficients ($S\tau_{b_{sig}}$)

Let $\tau_{b(m_1,m_2,p_i)}$ is the correlation coefficients for two metrics m_1 and m_2 in project p_i. Now we compute the mean for $S\tau_{b_{sig}}$ by the following equation:

$$\overline{\tau}_{b(m_1,m_2)} = \frac{1}{\nu}\sum_{i=0}^{q} \tau_{b(m_1,m_2,p_i)} \mid \tau_{b(m_1,m_2,p_i)} \in \tau_{b_{sig}} \tag{1}$$

Since (1) is calculated based on $S\tau_{b_{sig}}$, ν indicates the total count of $\tau_{b(m_1,m_2,p_i)}$ from all projects. To compute the sample mean, we only have to replace ν by m in (1). Since we are unsure of the τ_b values within set $S\tau_{b_{nsig}}$ due to their insignificant *p-value*, considering a conservative measure, we take τ_b from set $S\tau_{b_{nsig}}$ as zero. Thus, $\tau_{b(m_1,m_2,p_i)} \in S\tau_{b_{sig}}$ part in (1) still holds for the sample mean.

Like $S\tau_{b_{all}}$, the advantage of aggregating τ_b from the set $S\tau_{b_{sig}}$ is also that we can report the results using a single matrix without having the mentioned problems of $S\tau_{b_{all}}$ based aggregation. However, from such results, we are not able to determine whether $\overline{\tau}_{b(m_1,m_2)}$ is coming from a large or small value of ν. In other words, we are unable to tell how representative $\overline{\tau}_{b(m_1,m_2)}$ is among the selected projects. Since we can calculate sample mean from $S\tau_{b_{sig}}$, we can also compute variance and standard deviation to see the overall spread of τ_b within the samples. However, sample mean and standard deviation of τ_b from $S\tau_{b_{sig}}$ do not necessarily tell us about the distribution of τ_b within the four strength levels in Table 10. Standard deviation gives us an indication of the variability, but we do not have a way to know how the variability looks like regarding different strength levels of τ_b.

– Aggregating correlation coefficients (τ_b) based on the sets $S\tau_{b_{vs}}$, $S\tau_{b_s}$, $S\tau_{b_m}$, $S\tau_{b_w}$

  These four sets represent τ_b based on their strengths according to Table 10. These sets make $S\tau_{b_{sig}}$ i.e., $S\tau_{b_{vs}} \cup S\tau_{b_s} \cup S\tau_{b_m} \cup S\tau_{b_w} = S\tau_{b_{sig}}$. We can consider two measures from these sets as listed below.

  – *Count of* τ_b based on their level of strength: We get this measure by simply counting the number of τ_b within a set. This simple measure gives us a direct answer to the question, "how many projects report a certain correlation between two metrics at a certain level of strength"? Since we have 21 projects, the maximum value *count of* τ_b can be 21 and minimum can be zero. Looking at particular values of a *count of* τ_b for two metrics from the metrics $S\tau_{b_{nsig}}$, $S\tau_{b_{vs}}$, $S\tau_{b_s}$, $S\tau_{b_m}$, and $S\tau_{b_w}$, we can understand the distribution of τ_b among different sets toward having a better understanding about the nature of relationships between metrics.
  – *Sum of* τ_b based on their level of strength: Instead of taking counts, this adds all τ_b within the set. Since the highest value of a single τ_b is 1.0, the maximum value for *sum of* τ_b can also be 21 similar to the measure *count of* τ_b. However, *sum of* τ_b does not tell us how many τ_b values are contributing for the sum, which we can get from the *count of* τ_b. Thus, these two measures are mutually exclusive.

These two proposed measures, *count of* τ_b and *sum of* τ_b, provide additional insights about correlation coefficients compared to popular statistical measures sample mean and standard deviation. In this report, when we mention the term 'mean,' we indicate it for a certain set, and when data from all sets are considered, we write it as 'sample mean.' For example, when we say 'correlation between metrics m_1 and m_2 results in τ_b' or 'correlation between metrics m_1 and m_2 results in very strong/strong/moderate/weak τ_b', we refer to τ_b from the sample mean table.

4.3 Missing Correlation Coefficients (τ_b)

Correlation cannot be computed if either or both of the variables have constant values or missing values. In such a case, computation of correlation returns NaN (not a number) for both τ_b and *p-value*. Set S_{NaN} contains such data. S_{NaN} is kept disjoint from the $S\tau_{b_{all}}$ in Fig. 6, because τ_b value is missing to determine the level of strength and *p-value* is also missing to determine the level of significance. Even though S_{NaN} does not help us with correlation, it tells us about the nature of specific metrics.

When discussing and reporting results in the following section, we have to point to different sections of correlation matrices or derived matrices from them. To simplify the referencing, we label different sections of such matrices as shown in Table 11.

When reporting this case study, we have tried to maintain the actual flow how this research was carried out. Based on some interesting observations between inter-category correlations of 15 cumulative and three organic metrics, this case study further tested the hypothesis:

"*The median difference between correlations of metrics from cumulative and organic$_t$ categories equals to zero*."

This required deriving a set of 15 organic metrics denoted as organic$_t$ corresponding to the 15 cumulative metrics. The specifics of designing, performing and results of the test are elaborated in Section 5.4.

5 Results

Before going into the results and discussion regarding significant correlation coefficient (i.e., τ_b based on significant *p-value*), from the sets within $S\tau_{b_{sig}}$, we report how correlations coefficients outside $S\tau_{b_{sig}}$ looks like to illustrate data that is not contributing to the main result.

5.1 Missing and Non-significant Correlation Coefficients (τ_b)

First, we look at the small set S_{NaN} in Fig. 6, reporting cases where computing correlation was not successful and resulted in null values (i.e., NaN) for τ_b and *p-value* for specific pairs of metrics from M as reported in Table 19. Missing τ_b related to the metric *directories* as

Table 11 Labeling different sections of our matrices for easy referencing

		Cumulative			Density			Average			Organic		
		ncloc	classes	...	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
Cumulative	ncloc classes ...	Section CC			Section CD			Section CA			Section CO		
Density	comment-ln-den pub-doc-api-den dup_ln_dens	Section CD			Section DD			Section DA			Section DO		
Average	file_complexity class_cmplxt fn_cmplxt	Section CA			Section DA			Section AA			Section AO		
Organic	new_lines new_dup_ln new_du_block	Section CO			Section DO			Section AO			Section OO		

Table 12 Category-wise mean of *count of non-significant* correlation coefficients (τ_b) from Table 17

	Cumulative	Density	Average	Organic	Sum
Cumulative	0.27	2.49	2.44	11.56	16.76
Density	2.49	2.33	3.00	12.33	20.16
Average	2.44	3.00	1.67	13.67	20.78
Organic	11.56	12.33	13.67	0.00	37.56

seen in this table sourced from two projects, *malmo* and *geometry-api-java*. In both projects, the metric *directories* has values (8 for *malmo* and 3 for *geometry-api-java*) that remained unchanged throughout the project. The rest of the missing τ_b results from the project *cloud*, and all six metrics are related to duplications category that are *duplicated_lines*, *duplicated_blocks*, *duplicated_files*, *duplicated_lines_density*, *new_duplicated_lines*, and *new_duplicated_blocks*. Since the *cloud* project has no duplication related issues during the analyzed revisions, all six metrics have the value 0.

If at least one metric from a pair contains a constant value (i.e., a metric having the same value for all revisions), it is not possible to perform correlation on that pair of metrics, and this results in NaN values for τ_b and *p-value*. We observed the *directories* metric and metrics related to duplications have fewer levels (meaning variations) compared to other metrics.

Now, we move to the set $S\tau_{b_{nsig}}$ as reported in Table 17. Horizontal bars in this table and all other similar tables reporting the count and sum measures are graphical representations of the corresponding cell values. The maximum value for a cell corresponding to two metrics (e.g., the cell between *ncloc* and *classes*) is q which is 21, the number of projects. Cells in the principal diagonal are omitted, and thus, kept blank. The rightmost column reporting the 'total count' or 'total sum' of a row may have a maximum possible value of 483 (calculated from $n.q - q$). The horizontal bars are drawn based on the maximum possible value a cell can contain (i.e., 21 for regular cells between metrics and 483 for cells indicating 'total') but not on the maximum available value in a table.

A single glance at Table 17 gives us the impression that organic category is different from all other categories. A more careful look tells us that there are three groups: cumulative and organic having lowest and highest count of non-significant τ_b values correspondingly, and

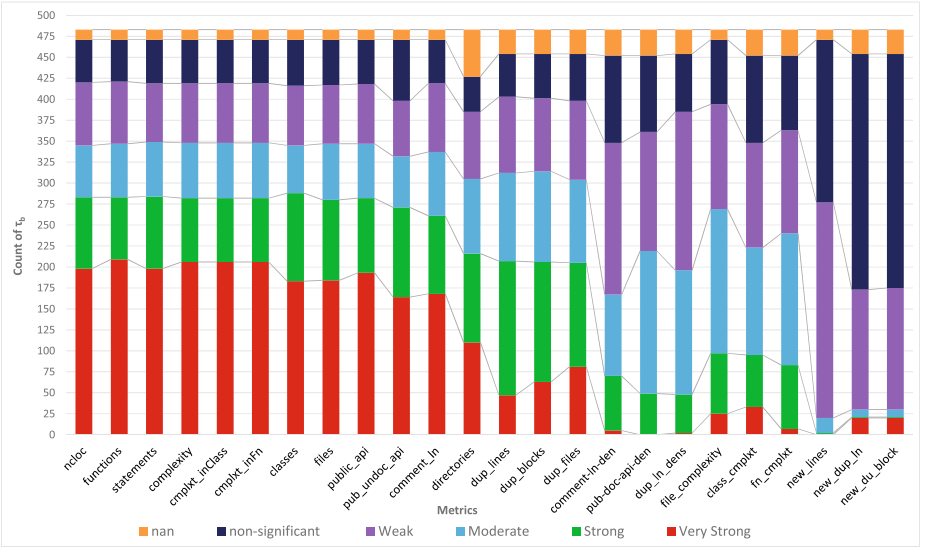


Fig. 7 The overall distribution of *count of* τ_b (correlation coefficients) at different levels with respect to all metrics

density and average with the count of non-significant τ_b values in between. We can also see it from the rightmost column 'total count'. Since 'total count' comes from all four metric categories, we show the category-wise mean of Tables 17 in 12.

We are interested to see the numbers in the principal diagonal of Table 12, which indicates measures coming from correlations between metrics from within a category, i.e., intra-category correlations. We see section OO has the least amount of non-significant τ_b followed by section CC, AA, and DD. Metrics within the organic category (section OO) are interesting as they have the least amount of non-significant τ_b within itself, however, organic scored highest when correlated to other categories. Another observation is that the mean value of 'count of non-significant τ_b' within a category itself is always smaller than the mean value of 'count of non-significant τ_b' between categories. Since we cannot draw any conclusion whether τ_b is strong or weak from τ_b with non-significant *p-value*, it is better to have less non-significant values within the context of making statistical analysis.

Key Observations:

- Correlating metrics from different categories results in more non-significant correlation coefficients compared to correlating metrics within a category.
- Category organic is quite different from the other three categories by producing a lot of non-significant τ_b for intra-category correlations.
- Category organic has least amount of non-significant τ_b followed by cumulative, average, and density for inter-category correlations. Even though the mean value for cumulative category is quite low (0.27) in this context, the contrast between the inter and intra-category is clearly less noticeable compared to organic category.

5.2 Overall Distribution of Correlation Coefficients (τ_b)

Based on the labeling of τ_b in Table 10, we have reported two sets of measurements. They are 'count of τ_b' (Tables 23, 24, 25, and 26) and 'sum of τ_b' (Tables 27, 28, 29, and 30) at different levels.

Figure 7 is constructed based on the rightmost column 'total count' from Tables 17, 19 23, 24, 25, and 26. Similarly, Fig. 8 is constructed from the absolute values of the rightmost column 'total sum' from Tables 27, 28, 29, and 30. Since 'total count/' and 'total sum' columns in these

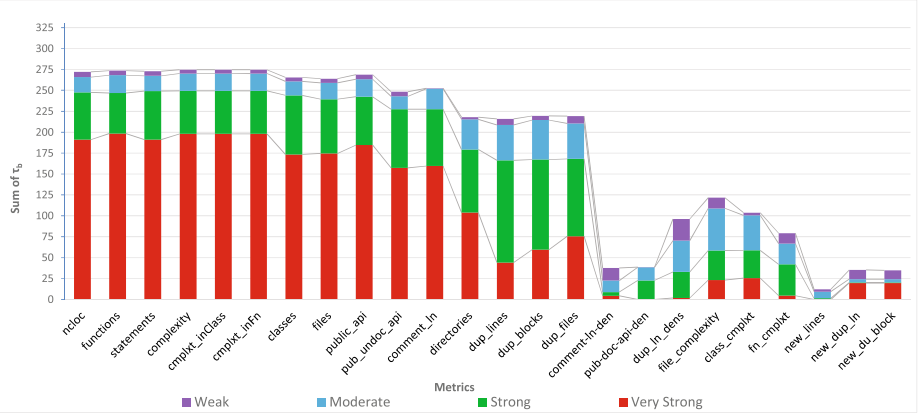


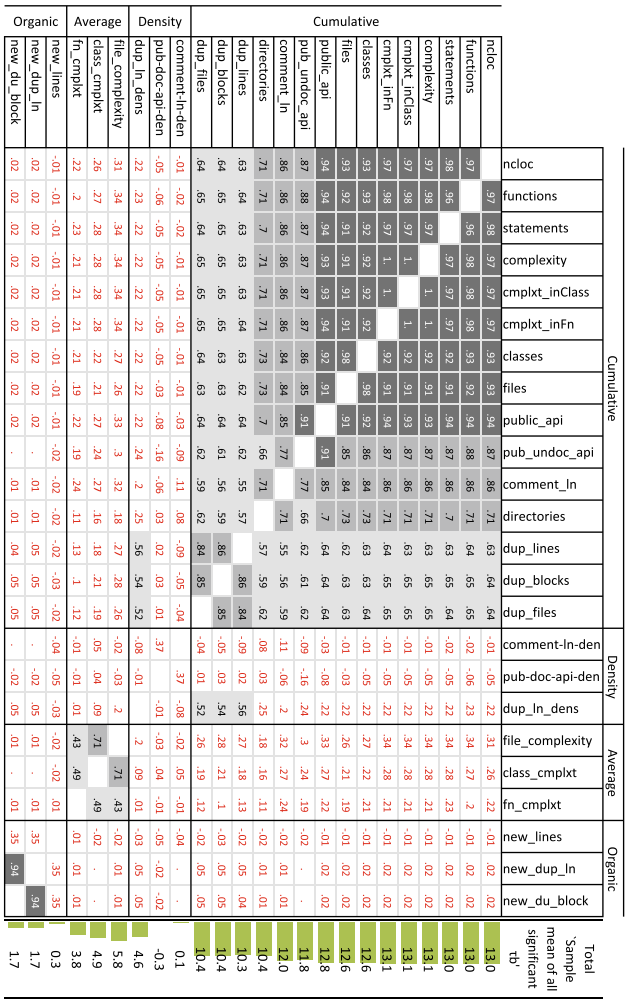
Fig. 8 The overall distribution of *sum of* τ_b (correlation coefficients) at different levels with respect to all metrics

tables are counts and sums of corresponding measures of a metric's correlation with respect to all other metrics, we get an overall distribution of τ_b for the metrics from these columns.

Both Figs. 7 and 8 are vertically equally scaled for better comparison. Between these figures, the level 'very strong' has the least difference and level 'weak' has the highest difference among the four strength levels of τ_b. While Fig. 7 gives us an overview of how different sets of τ_b are representative, Fig. 8 shows the actual sum of τ_b. Since for $S\tau_{b_{nsig}}$ τ_b value is meaningless, and for S_{NaN} τ_b value is missing; they are not included in Fig. 8.

In Fig. 8, we can see how the red bars, representing 'very strong τ_b' within the cumulative metrics (metrics *ncloc* to *duplicated_files*), are dominating compared to other levels. Metrics *directories*, *duplicated_lines*, *duplicated_blocks*, and *duplicated_files* are comparatively weaker in terms of 'very strong τ_b' compared to the other cumulative metrics. However, all cumulative metrics have scored much higher than metrics from other measurement categories. Metrics from the organic category have scored lowest among all metrics and categories. These two figures represent data from all correlations, e.g., the horizontal bar

Table 13 Sample mean of significant correlation coefficients (τ_b) from the set $S\tau_{b_{all}}$ (set of all τ_b). The three gray-scale cell colors indicate three levels of τ_b. Cells with red text indicate weak τ_b



for *ncloc* comes from the correlation coefficient of *ncloc* with all other metrics. Thus, from these figures, we cannot determine whether there is any difference between τ_b's resulting from intra-category metrics correlation and inter-category metric correlation. We study this in more detail next.

5.3 Significant Correlation Coefficients

All reported τ_b from this subsection passed the significance $\alpha = 0.05$, which we will not mention any further for the rest of this subsection. When reporting τ_b, by default, we indicate the sample mean as reported in Table 13.

First, we want to look at intra-category relations among the metrics. Table 13 shows the sample mean of τ_b corresponding to the set $S\tau_{b_{all}}$, and Table 21 in the Appendix shows the mean of τ_b within the $S\tau_{b_{sig}}$. Here, we want to reiterate that we have considered any non-significant τ_b from the set $S\tau_{b_{all}}$ as 0.

Perfect Correlations We are interested in the perfect correlation (i.e., $\tau_b = 1.0$) reported in Tables 13 and 21 between metrics (*complexity*, *complexity_in_classes*), (*complexity*, *complexity_in_functions*), and (*complexity_in_classes*, *complexity_in _functions*) since this is an indication that these three pairs of metrics are perfectly correlated meaning both metrics within a pair measure exactly the same aspect. It can be noted that τ_b for these relations are not exactly 1 as reported in Tables 13 and 21. This happens because we have reported τ_b up to two decimal points, so anything greater or equal to 0.995 is reported as 1. To understand these relations, we counted the perfect correlation coefficients between all metrics from all projects, which is reported in Table 20 where we see five relations have perfect τ_b. In 20 projects the correlation between *complexity* and *complexity_in_classes* is perfect. So it is evident that the metrics *complexity* and *complexity_in_classes* are measuring the same aspect and using one of them is sufficient. Since our data is coming from Java source code, this is not a surprise because, in Java, code does not reside outside classes.

For the relation between *complexity* and *complexity_in_functions*, there exist perfect correlations in nine projects. Since we found a perfect correlation in the sample mean of τ_b in Table 13, it means the rest of the 12 τ_b for this relation must be very strong. We have also calculated 'sum of significant τ_b' reported in Table 22. The 'sum of significant τ_b' for (*complexity*, *complexity_in_functions*) is found to be 20.98 out of 21. This indicates that *complexity* and *complexity_in_functions* also measure the same aspect with a negligible difference.

Even though the results in Table 20 does not show any perfect correlation between the metrics *ncloc*, *functions*, *statements*, *complexity*, *classes*, *files*, and *public_api*, but we see τ_b greater than 0.9 meaning at the very strong level for all relations in the sample mean in Table 13. Considering any relation at the τ_b-level greater or equal to 0.9 redundant, we consider all these seven measures from the cumulative category as redundant. Under the same consideration, *public_api* is redundant to *public_undocumented_api*.

For relations between *new_duplicated_lines* and *new_duplicated_blocks* and between *duplicated_blocks* and *duplicated_files*, perfect correlations, are found in only one project. 'Sum of significant τ_b' for these two pairs are reported in Table 22 as 19.65 and 17.87 correspondingly. Thus, we cannot say right away that metrics within these two pairs are duplicated.

5.3.1 Intra-Category Correlations

Intra-category correlations indicate correlations within the sections CC, DD, AA, and OO where all correlations within a section are coming from metrics measured similarly.

Table 14 Category-wise mean of sample mean of significant correlation coefficients (τ_b) from Table 13

	Cumulative	Density	Average	Organic	Sum
Cumulative	0.79	0.08	0.24	0.01	1.12
Density	0.08	0.09	0.03	-0.01	0.20
Average	0.24	0.03	0.54	0.00	0.82
Organic	0.01	-0.01	0.00	0.55	0.55

Section *CC* The sample mean of $S\tau b_{all}$ in Table 13 presents that metrics within the cumulative category (i.e., section CC) are very highly correlated to each other, which we can see from the category-wise mean of the sample mean values in Table 14. Here, section CC has a mean value of 0.79 for τ_b which indicates a strong correlation coefficient between any metrics for any projects. For a more detailed picture, we look at the sample mean values in Table 13 where we can see metrics are mainly divided into three parts based on the strength of correlations. The first part constitutes of nine redundant metrics (from *ncloc* to *public_api*), which we have discussed earlier, i.e., due to 'very strong' τ_b, any of these metrics can explain the variability of the other two metrics to a high degree. The second part has three metrics *public_undocumented_api*, *comment_lines*, and *directories* that are mostly within the τ_b level 'strong', except *public_undocumented_api* is redundant to *public_api* (as discussed in the preceding section), and the correlation between *public_undocumented_api* and *directories* is 'medium'. The third part has *duplicated_lines*, *duplicated_blocks*, and *duplicated_files* that are all 'strongly' correlated to each other but they are correlated at a 'medium' level with all the metrics from the other two parts.

Since the sample mean (i.e., from $S\tau b_{all}$) reported in Table 13 is a more conservative measure than the mean of $S\tau b_{sig}$ in Table 21, we expect equal or better result (higher τ_b value) in Table 21. From the data, we see a very small or no difference for most of the mentioned metrics due to the few numbers of non-significant and missing τ_b in section CC.

Key Observations:

- Metrics *complexity*, *complexity_in_classes*, and *complexity_in_functions* measure exactly the same aspect.
- Metrics *ncloc*, *functions*, *statements*, *complexity*, *classes*, *files*, *public_api*, and including the three metrics mentioned above are all redundant.
- Metrics *public_api* and *public_undocumented_api* are redundant.
- Not a single correlation out of 105 total correlations within the cumulative metrics has a weak correlation coefficient.

Sections *DD*, *AA*, *OO* As Table 14 shows, sections DD, AA, and OO have on average weak to moderate correlations compared to strong correlations in section CC. Correlations among all three metrics (*comment_lines_density*, *public_documented_api_density*, and *duplicated_lines_density*) from density category in section DD have weak τ_b. For section AA, the correlation between *file_complexity* and *class_complexity* from average category has a strong τ_b of 0.71. The other two correlations (including *function_complexity*) for this section have moderate τ_b. In section OO, the correlation between *new_duplicated_lines* and *new_duplicated_blocks* is 0.94 and, based on our 0.9 limit; these two metrics are redundant. Correlations of these two metrics with *new_lines* are weak.

All four sections (related to intra-category) have less non-significant τ_b (in Table 12) compared to other sections, and section OO has no non-significant τ_b. Section OO also has the lowest category-wise mean of standard deviations of τ_b as reported in Table 15. The individual records of standard deviations in Table 18 show that redundant metrics within section

Table 15 Category-wise mean of standard deviations of significant correlation coefficients (τ_b) from Table 18

	Cumulative	Density	Average	Organic	Sum
Cumulative	0.19	0.47	0.52	0.18	1.36
Density	0.47	0.40	0.45	0.15	1.47
Average	0.52	0.45	0.27	0.20	1.43
Organic	0.18	0.15	0.20	0.09	0.61

CC have lower standard deviations than other metrics from the same category. Density metrics in section DD have the highest standard deviation and we have to look into 'count of τ_b' (Tables 23, 24, 25 and 26) and 'sum of τ_b' (Tables 27, 28, 29 and 30) tables to fully understand how τ_b values are distributed in this category.

Intra-category correlations among cumulative metrics are much higher than density, average, and organic. Even though cumulative, average, and organic are all at the strong τ_b level, but for cumulative the category-wise mean of the sample mean of τ_b is 0.79, thus, almost close to very strong level. On the other hand, average and organic categories have scored 0.54 and 0.55 correspondingly. In addition, even though we do not see a single weak level correlation in CC, more than half (5 out of 9) correlations in sections DD, AA, and OO have weak τ_b.

Key Observations:

- Metrics *new_duplicated_lines* and *new_duplicated_blocks* (in section OO) are redundant.
- All correlations among density metrics are weak.
- Intra-category correlations for density, average, and organic metrics result in lower τ_b compared to cumulative metrics.

5.3.2 Inter-Category Correlations

Inter-category correlation happens on metrics that are measured differently. We want to see whether there exist any noticeable difference in inter-category correlations compared to intra-category correlations.

Inter-category correlations are available in the six sections that are CD, CA, DA, CO, DO, and AO. These sections reflect all possible correlations, a total of 162, between metrics from all four categories. All these correlations result in weak τ_b except three correlations that result in moderate τ_b (values 0.56, 0.54, and 0.52) as shown in Table 13. Having a look at the category-wise mean in Table 14, we see that sections CO, DO, and AO, i.e., inter-category correlations of organic metrics with all other categories are the lowest. Interestingly, standard deviations are also lower for these three sections related to organic category (in Tables 18 and 15), meaning when organic metrics are correlated with metrics from other categories, the variability of τ_b is low. When we look at the *count of* τ_b (Tables 23, 24, 25, and 26) and *sum of* τ_b (Tables 27, 28, 29, and 30), we see that sections CO, DO, and AO have values only in Tables 26 and 30 reporting weak τ_b. In the remaining tables these three sections have zero. For other three sections (CD, CA, and DA), we see 'count of τ_b' and 'sum of τ_b' measures are available in all four levels and most concentrated at the moderate level.

When we look at the difference between intra- and inter-category correlation of metrics, it is clear that they are different. The grand mean of the category-wise mean (see Table 14) of intra-category correlations is 0.49 (from $(0.79 + 0.09 + 0.54 + 0.55)/4$), however, for inter-category correlations, it is 0.06 (from $(0.08 + 0.24 + 0.01 + 0.03 - 0.01 + 0)/6$).

The observation that correlation between metrics from different categories results in overall weak correlation is important because this tells us that metrics from different categories have low collinearity. Thus, software code metrics from different categories can be used together as features in models for prediction, forecast, etc.

Key Observations:

- Intra-category correlations of metrics are different from inter-category correlations by resulting in much lower correlation coefficients.
- Correlation of metrics from different categories results in overall weak correlation coefficient. Thus, code metrics from different categories are observed to have low correlation coefficients, thus, are non-collinear.

Overall, we have observed that cumulative metrics are different because the intra-category correlations of cumulative metrics result in higher τ_b values compared to correlations of metrics within other categories.

5.4 Cumulative vs. Organic Metrics

The progression of development of software can be tracked in different ways. Traditionally, we keep track of software through cumulative metrics. Cumulative metrics are intuitive in the sense that they give us an overall idea of the state of the system. The same thing, i.e., tracking the progression of software, can also be done through the organic way. Version control systems like Git keep track of revisions through saving deltas from each revision. Taking the delta from each consecutive software revisions, we can still calculate the total by a linear addition. If we have either a cumulative or an organic measure, we can calculate one from the other.

Now we have a question whether the higher τ_b values among cumulative metrics occur due to the cumulative way of measurement or it is just because the cumulative metrics are more correlated to each other. If we want to test this, we have to transform all the cumulative metrics in this study into organic metrics (which we denote as organic$_t$) then compute correlations and check whether there is a significant difference or not. It can be noted that in Table 2, the three organic metrics are different from the 15 cumulative metrics. Thus, we need to create a new set of organic metrics equivalent to the cumulative metrics. Since we have identified few perfectly correlated cumulative metrics, they can act as our point of validation. Meaning, perfectly correlated metrics should always be perfectly correlated no matter how they are measured. To check the difference between cumulative and organic$_t$ categories, we have considered the following hypothesis:

- **Null hypothesis** H_0: The median difference between correlations of metrics from cumulative and organic$_t$ categories equals to zero.

5.4.1 Transformed Organic Metrics

We have transformed all 15 cumulative metrics into organic$_t$ metrics by taking the difference between consecutive revisions for each metric. We name these new metrics by adding an underscore before the cumulative counterpart metrics from which they are transformed, e.g., *ncloc* is transformed into *_ncloc*. It should be noted that the existing three metrics from the organic category do not take any negative values. Meaning if *ncloc* decreases *new_lines* will hold a zero value because there are no new lines. However, organic$_t$ metrics can hold negative values reflecting a reduction of a metric's measure in consecutive revisions.

5.4.2 Designing and Executing the Test

After computing the organic$_t$ metrics, we went through the same procedure as we did for the other metrics in this study, i.e., checking the nature of the data. We found organic$_t$ metrics to be similar to organic metrics in terms of kurtosis and short tails. However, for skewness,

Table 16 Sample mean values for significant correlation coefficients (τ_b) of all metrics including transformed organic metrics. Three gray-scale cell colors indicate three levels of τ_b. Cells with red text indicate weak τ_b. A dot (.) in a cell indicates zero value

organic$_t$ metrics are not as extremely left skewed as organic metrics because organic$_t$ metrics can take negative values. However, overall, organic$_t$ metrics are non-normal.

We performed Kendall's τ for all 39 metrics and derived tables similar to the tables mentioned earlier in Section 5. The sample mean of the significant correlation coefficients is presented in full in Table 16. However, in the interest of space, we only reported newly derived table sections for standard deviation, count of non-significant τ_b, and count of perfect correlations in Tables 31 and 32.

5.4.3 Results from the Test

In Tables 16, 31, and 32, we see that organic$_t$ metrics are similar to organic metrics both in terms of intra- and inter-category correlations.

Now we like to see whether organic$_t$ metrics are able to produce perfect correlations as produced by the cumulative metrics. Following the similar referencing style as per Table 11, we refer the section containing τ_b from intra-category correlations of organic$_t$ metrics as $O_t O_t$. Intra-category correlation of cumulative metrics (in Section CC of Table 20) have a total 46 counts of perfect correlations for six correlations. For organic$_t$ metrics, we see (in section $O_t O_t$ of Table 32) 24 perfect correlations for four metrics are exactly produced. However, for two correlations ((*complexity*, *complexity_in_functions*) and (*complexity_in_classes*, *complexity_in_functions*)) 12 out of 22 perfect correlations are produced. So we look at the sample mean of these two correlations (see section $O_t O_t$ of Table 16) and find a value 0.994 for both of these correlations; the value 0.994 is so close to the maximum possible τ_b value of 1 that we can consider 0.994 as a perfect correlation. Thus, we see that both cumulative and organic$_t$ metrics are equally able to detect perfect correlations among metrics if there exist any.

At this point, we like to focus on testing the null hypothesis. The mean of section $O_t O_t$ of Table 16 is calculated as 0.49 which is much lower than the value of 0.79 for the cumulative metrics (in section CC of Table 14). However, without performing a statistical test, we cannot accept or reject our null hypothesis.

Earlier our data was the code metrics, but now it is the sample mean of the Kendall's τ. So now we need to check the distributions of data to be tested, i.e., section CC and $O_t O_t$ of Table 16. We have checked descriptive statistics, skewness, kurtosis, histograms, and also performed the Shapiro-Wilk test on these two data sets and found the data non-normal. Thus, we have to choose non-parametric tests.

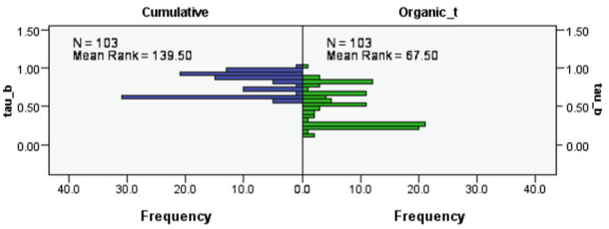
We can consider the data as paired because a single τ_b value, say *ncloc* in section CC and a corresponding τ_b value from section $O_t O_t$ (i.e., *_ncloc*) both measure the similar aspect but they are measured differently. Then it can also be argued that *ncloc* and *_ncloc* are two different measures and they cannot be considered as paired. Taking both arguments, we would like to execute two tests to see whether there is any significant difference. Taking our two data sets as two independent samples, we perform the 'Mann-Whitney U Test', and taking our data sets as dependent samples, we perform the 'Wilcoxon Signed Ranks Test'. After checking the assumptions of these tests, we remove two τ_b from section CC and the two corresponding τ_b from section $O_t O_t$. Since the correlations between metrics *complexity*, *complexity_in_classes*, *complexity_in_functions* in section CC and corresponding organic$_t$ measures from section $O_t O_t$ are commonly identified as perfectly correlated in both sections, we decide to keep only one of them and remove the other two so that the assumption related to data dependency is no longer present. The 'Wilcoxon Signed Ranks Test' has an assumption that all the difference between paired data should approximately be equally distributed along the quartiles when plotted as a one-dimensional box-plot. This was

Test Statistics*

	τ_b
Mann-Whitney U	1596.500
Z	-8.668
Asymp. Sig. (2-tailed)	.000

*. Grouping Variable: (Cumulative, Organic$_t$)

(a) Mann-Whitney U Test



(b) Distributions for two data sets

Fig. 9 Statistical test considering cumulative and organic$_t$ as two independent samples. Sub-figure 9a shows test statistics for 'Mann-Whitney U Test' and 9b graphically shows distributions of τ_b values for metrics within cumulative and organic$_t$ categories

not met for our data. Since this assumption is a visual test, we decided to include a 'Paired-Samples Sign Test', which does not have such an assumption. We report results from all these three tests here.

We have a total 103 pairs. From the test ranks, we see that for both 'Wilcoxon Signed Ranks Test' and 'Paired-Samples Sign Test' count of negative difference is 0, count of positive difference is 102, and count of ties is one, taking CC as the first and $O_t O_t$ as the second variable. These numbers already tell us without looking at the significance levels, that the organic$_t$ category is lower than the cumulative category. From Figs. 9 and 10, we see that all the three tests show a *p-value* less than 0.01 (i.e., considering an $\alpha = 0.01$). Thus, we can reject the null hypothesis and accept the alternative hypothesis, i.e., the median difference between correlations of metrics from cumulative and organic$_t$ categories is not equal to zero.

5.4.4 Implications of the Test Results

The finding that there exists a significant median difference between correlations of metrics from cumulative and organic categories, implies that organic metrics are a set of measures that are collectively different than the cumulative metrics. Therefore, organic metrics can be considered as a new set of feature holding different characteristics than cumulative metrics as a whole.

The intra-category correlations of cumulative metrics are much higher than their equivalent sets of transformed organic metrics. Since correlations between cumulative metrics are high, there exist high collinearity among these metrics. This makes cumulative metrics collinear, and only a single metric from a group of highly correlated metrics can be considered as a valid input feature for a predictive model. The high collinearity among software code metrics is not new information. However, the knowledge that transforming cumulative metrics into organic can significantly reduce the collinearity is new. Since this

	Cumulative-Organic$_t$
Z	-8.768*
Asymp. Significance (2-tailed)	.000

*. Based on negative ranks.

(a) Wilcoxon Signed Ranks Test

	Cumulative-Organic$_t$
Z	-10.000
Asymp. Significance (2-tailed)	.000

(b) Paired-Samples Sign Test

Fig. 10 Statistics for different statistical tests to determine the existence of significant differences between correlations from cumulative and organic$_t$

transformation does not alter the original footprint of how software is evolved, it is expected to be free of side effects of normalization. Since organic metrics have lower collinearity, the chance of having multiple valid input features from them is possible.

The inter-category correlations of metrics from different categories are observed to be always low. This information can improve feature engineering by making the process more systematic. First, this gives a simple and intuitive understanding of non-collinearity based on measurement types. Second, this implies transforming a feature into a different measurement type can produce a feature that is non-collinear with the original feature.

5.5 Discussion

Software engineering researchers have observed high collinearity among software code metrics. However, we did not have explicit knowledge whether the high collinearity is due to the inherent nature of the code metrics or due to how we measure them. This study compares between correlation coefficients of a set of 15 cumulative metrics and correlation coefficients of their corresponding organic metrics and reveals that a large portion of collinearity among cumulative metrics results from the cumulative way of measurement. Since organic metrics are free from the effects of cumulation representing the natural evolution of software, we think, correlation coefficients among the organic metrics represent the inherent collinearity among code metrics. Taking the difference between the intra-correlation coefficients of metrics from these two categories, we can determine the added collinearity due to cumulative measurement.

High collinearity among a group of input features makes them weaker as a whole. We can get a few valid input features from such a set because, during the validation process, features with high collinearity are removed. Since organic metrics have lower collinearity among themselves, we can possibly get more valid input features from them. Moreover, since the collinearity between cumulative and organic metrics are very low, we can combine them to have even more valid input features. The lower collinearity among cumulative and organic metrics also means, even though we can calculate a set of organic metrics corresponding to a set of cumulative metrics, they are mutually exclusive. Therefore, theoretically, we do not have to restrict our choice of metrics from either of these two categories.

It is interesting that when we add density and average categories to the scenario described above for cumulative and organic, still non-collinearity holds between metrics from different measurement categories. It can be noted that the unit of measurement of the density metrics is percentage. While cumulative and organic metrics have the same value type (i.e., integer), metrics are measured differently in both categories.

The findings that organic metrics are collectively different than their corresponding cumulative metrics (i.e., inter-category), the understanding of the effect of cumulation toward collinearity among metrics (i.e., intra-category), and the general observation that metrics from different measurement categories yield in overall weak correlation coefficients are significant for the researchers and the practitioners in software engineering because they help us better understand the nature of the code metrics. The findings open the possibility of new research and revisiting existing research in this field. Based on our results, theoretically, we have more valid input features from different measurement categories, however, in practice, research needs to be conducted to determine which predictors (i.e., input metrics) are good for which targets (i.e., qualities that we want to predict or estimate). It could be the case that specific metrics from a category work better in combination with other metrics to predict a particular quality attribute. Researchers can also try to find new measurement categories and their properties. This study has investigated some code metrics, other code metrics not covered here can also be studied. These results can help the practitioners by

giving them more insights about software metrics. Quality managers can re-prioritize the metrics that a project keeps track. Tool developers can rethink about the metrics their tools support. For example, SonarQube automatically removes *new_duplicated_lines*, *new_lines*, *new_duplicated_blocks*, and some other similar organic metrics. Based on the findings of this study, they can decide to give users the option to keep track of such metrics. Software engineers working with predictive analysis have more options when choosing input features for their models. For example, for a predictive model, *complexity* (of type cumulative) is identified as an input feature. At this point, the possibility of incorporating metrics related to *complexity* from different measurement categories can be explored and *complexity* per file (of type average) and *complexity* per commit or based on a specific time duration (of type organic) can be considered as input features.

5.6 Threats to Validity and Limitations

5.6.1 Internal Validity

Today software is built with various languages and it is very common that a project uses code from different languages. Still, a project is usually designated with a specific language indicating the major portion of code, etc. Computer programming languages use different constructs and measures of code metrics may dramatically vary due to language difference. To eliminate this effect, we choose to focus on a single programming language, Java. It can be noted that Java projects are among the top three ranks on GitHub.

We considered some other factors when selecting projects such as project types, project size in LOC, number of revisions, number of developers. While selecting the projects, we tried to combine these factors so that we have a good representation regarding these factors. Besides, we looked at the number of issues and pull requests. We tried to select projects that have reported issues and pull requests because these factors are signs of active involvements of users and developers.

Tools measuring software code metrics are not perfect. Different tools implement measures differently even though they claim to measure a same aspect of code (Lincke et al. 2008). This study has selected one of the most widely used measurement tools for software quality, and we have observed very strong correlations between the cumulative metrics similar to the major studies. However, selecting a popular tool and similar observations to the major studies do not entirely remove this threat. This is a general issue to any study like this and we are aware of this.

We mentioned earlier that we only checked the revisions from the master branch of a Git repository. This affects granularity of the collected data. Based on the finding in this study, we know that reducing the cumulative effect reduces the correlation between metrics, thus, avoiding partial cumulation of data due to Git branches could possibly make the result of this study more stronger. Therefore, we do not consider this a major threat to our results.

5.6.2 External Validity

There are millions of software projects hosted on GitHub. Generalizing result for such a large population is a key validity threat for any study and we also identified generalizability as a considerable threat to validity. Selection of project is one of the most important things to minimize this threat. We tried to carefully select projects from well-known organizations to mitigate this risk. For example, we have included projects from Apache, a pioneering organization in the open source community, Microsoft, and other organizations with diverse

portfolios. On the positive side, we have found highly significant results from different tests. For example, Mann-Whitney U Test in Fig. 9a, Wilcoxon Signed Ranks Test in Fig. 10a, and Paired-Sample Sign Test in Fig. 10b have significance values $4.4e{-}18$, $1.8e{-}18$, and $1.5e{-}23$ correspondingly.

To safeguard the internal validity, we choose to restrict our focus to Java source code. This choice, however, seems to affect the external validity, meaning 'do the results hold for source code written in other programming languages?'. Since measurement types discussed in this study are independent of the programming languages, such a threat is not highly significant, in our opinion. However, more research can be done to be certain.

6 Conclusions

This empirical research investigates whether measurement types of software code metrics have an effect on their correlations. Through collecting and analyzing 24 code metrics from 11,874 revisions from 21 open source Java projects, we have found that measurement types have an effect on correlations. Analysis of data shows that 10 out of total 15 metrics that are measured cumulatively are redundant based on our criteria two metrics with a correlation coefficient of 0.9 or above are redundant. When the cumulative effect of these metrics is removed by transforming these 15 metrics into type organic, only three of them are identified as redundant. These three metrics are identified as perfectly correlated (i.e., they measure exactly the same aspect) in both categories, implying if metrics are truly correlated, correlations of their organic measures are able to identify it. In addition, our analysis shows that organic metrics result in significantly lower correlation coefficients compared to cumulative metrics for intra-category correlations. Furthermore, while some software metrics are closely related to each other resulting in high correlation coefficients to an extent to be considered redundant, many higher correlation coefficient values are due to measuring these metrics cumulatively. In other words, we should not be surprised seeing higher correlation coefficients for cumulative metrics, and we should be aware that measuring metrics by their natural development, i.e., organically result in much lower correlation coefficient.

Another interesting finding is correlations between metrics from different categories yield in overall weak correlation coefficients. This finding is important because metrics from cumulative, density, average, and organic categories can be combined together as features for predictive models. From another view point, this could improve the process of feature engineering by providing the information that transforming a feature into a different measurement type produces a new feature that is non-collinear with the original feature.

We have discussed why Kendall's τ version B fits more for software projects. We also discussed the landscape of correlation coefficients outlining possible sets considering both correlation coefficient and *p-value* which can be helpful for this type of study.

This study has attempted to reveal the fundamental relationships between measurement types and correlations of software code metrics. More evidence is required to generalize and extend this knowledge. Thus, replicated studies can be conducted considering various metrics, measurement tools, programming languages, and project types.

Acknowledgments The authors like to thank Dr. Aila Särkkä for her comments about some of the statistical analysis performed in this study and the anonymous reviewers for their valuable comments and suggestions that have significantly improved the clarity of this paper.

Appendix

Here are the tables that are not put in the paper to enhance the reading experience.

Table 17 Count of non-significant correlation coefficients (τ_b) from $S\tau_{b_{nsig}}$ (set of non-significant τ_b), where corresponding *p-values* do not satisfy the level of significance $\alpha = 0.05$

	Cumulative																						Density			Average			Organic			
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total Count of non-significant τ_b
ncloc		0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	2	2	2	4	2	9	13	13	51
functions	0		0	0	0	0	0	0	0	1	0	0	0	0	0	3	2	2	1	4	2	9	13	13	50
statements	0	0		0	0	0	0	0	0	1	0	0	0	0	0	4	2	2	2	5	2	9	13	13	52
complexity	0	0	0		0	0	0	0	0	1	0	0	0	0	0	4	2	2	1	4	2	9	13	13	52
cmplxt_inClass	0	0	0	0		0	0	0	0	1	0	0	0	0	0	4	2	2	1	4	3	9	13	13	52
cmplxt_inFn	0	0	0	0	0		0	0	0	1	0	0	0	0	0	4	2	2	1	4	3	9	13	13	52
classes	0	0	0	0	0	0		0	0	1	0	0	0	0	0	4	2	2	1	4	3	9	13	13	52
files	0	0	0	0	0	0	0		0	1	0	0	0	0	0	4	2	2	1	4	3	9	13	13	52
public_api	0	0	0	0	0	0	0	0		1	0	0	0	0	0	4	2	2	1	4	3	9	13	13	52
pub_undoc_api	1	1	1	1	1	1	1	1	1		1	1	1	1	2	3	2	3	3	5	3	10	14	14	73
comment_ln	0	0	0	0	0	0	0	0	0	1		0	0	0	1	3	1	2	1	4	2	8	13	13	52
directories	0	0	0	0	0	0	0	0	0	1	0		0	0	1	3	1	1	3	3	1	7	11	11	42
dup_lines	0	0	0	0	0	0	0	0	0	1	0	0		0	1	3	1	1	0	2	2	9	13	12	51
dup_blocks	0	0	0	0	0	0	0	0	0	1	0	0	0		0	3	1	1	1	1	2	9	13	12	53
dup_files	0	0	0	0	0	0	0	0	0	1	0	0	0	0		3	1	2	1	4	2	9	13	13	53
comment-ln-den	3	3	4	4	4	4	3	3	3	3	3	3	3	3	3		2	3	4	4	3	10	16	16	104
pub-doc-api-den	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	2		2	2	2	2	7	15	15	91
dup_ln_dens	2	2	2	2	2	2	2	2	2	3	2	1	1	1	2	3	2		2	2	3	11	15	15	91
file_complexity	2	1	1	1	1	1	4	3	2	3	2	3	3	2	3	4	2	2		4	3	11	14	14	69
class_cmplxt	4	4	5	4	4	4	4	4	4	5	4	3	2	1	3	5	4	2	4		1	11	15	15	77
fn_cmplxt	2	2	2	3	3	3	1	2	2	3	2	1	1	1	2	3	2	1	3	1		11	16	16	89
new_lines	9	9	9	9	9	9	9	8	9	10	8	7	9	10	10	9	7	11	11	11	11		0	0	194
new_dup_ln	13	13	13	13	13	13	13	14	13	14	13	11	13	13	12	16	15	15	14	15	16	0		0	281
new_du_block	13	13	13	13	13	13	13	14	13	14	13	11	12	12	12	16	15	11	14	15	16	0	0		279

Table 18 Standard deviations of significant τ_b from $S\tau_{b_{all}}$ (set of all τ_b)

		Cumulative																					Density			Average			Organic		
		ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
Cumulative	ncloc	.04	.04	.03	.04	.04	.04	.05	.07	.08	.13	.28	.29	.28	.27	.29	.54	.56	.37	.51	.55	.56	.19	.18	.18
	functions	.04	.04	.03	.04	.02	.01	.08	.1	.06	.11	.28	.29	.27	.27	.29	.55	.55	.35	.5	.55	.57	.19	.18	.18
	statements	.03	.04	.04	.02	.02	.04	.07	.09	.06	.12	.28	.29	.27	.27	.28	.55	.54	.36	.49	.54	.55	.19	.18	.18
	complexity	.04	.02	.04	.04	.	.	.08	.1	.06	.12	.28	.29	.27	.27	.28	.57	.55	.35	.49	.54	.57	.19	.18	.18
	cmplxt_inClass	.04	.02	.04	.	.04	.	.08	.1	.06	.12	.28	.29	.27	.27	.28	.57	.55	.35	.49	.54	.57	.19	.18	.18
	cmplxt_inFn	.04	.01	.04	.	.	.04	.08	.1	.06	.11	.28	.29	.27	.27	.28	.57	.55	.36	.5	.54	.57	.19	.18	.18
	classes	.05	.08	.07	.08	.08	.08	.08	.03	.11	.16	.29	.32	.27	.27	.28	.54	.59	.38	.54	.57	.55	.19	.18	.18
	files	.07	.1	.09	.1	.1	.1	.03	.03	.12	.17	.29	.32	.27	.28	.29	.53	.56	.38	.54	.57	.57	.18	.18	.18
	public_api	.08	.06	.06	.06	.06	.06	.06	.12	.12	.06	.29	.32	.26	.25	.27	.54	.57	.36	.5	.54	.56	.2	.17	.18
	pub_undoc_api	.13	.11	.12	.12	.12	.11	.16	.17	.06	.06	.32	.34	.27	.25	.27	.55	.57	.35	.51	.54	.57	.19	.16	.16
	comment_ln	.28	.28	.28	.28	.28	.28	.29	.29	.29	.32	.32	.31	.36	.36	.29	.53	.53	.37	.49	.54	.55	.19	.17	.18
	directories	.29	.29	.29	.29	.29	.29	.32	.32	.32	.34	.31	.24	.24	.18	.2	.48	.53	.34	.53	.54	.55	.2	.18	.18
	dup_lines	.28	.27	.27	.27	.27	.27	.27	.27	.26	.27	.36	.24	.08	.08	.13	.44	.55	.3	.44	.47	.5	.16	.14	.17
	dup_blocks	.28	.27	.27	.27	.27	.27	.27	.28	.25	.25	.36	.18	.08	.12	.12	.43	.54	.32	.42	.43	.49	.17	.14	.18
	dup_files	.29	.28	.28	.28	.28	.28	.28	.29	.27	.27	.29	.2	.13	.12	.12	.44	.55	.31	.44	.47	.51	.19	.18	.18
Density	comment-ln-den	.54	.55	.55	.57	.57	.57	.54	.53	.54	.55	.53	.48	.44	.43	.44		.42	.34	.42	.39	.45	.15	.14	.14
	pub-doc-api-den	.56	.55	.54	.55	.55	.55	.59	.56	.57	.57	.53	.53	.55	.54	.55	.42		.45	.45	.42	.52	.16	.16	.16
	dup_ln_dens	.37	.35	.36	.35	.35	.36	.38	.38	.36	.35	.37	.34	.3	.32	.31	.34	.45		.48	.44	.46	.13	.14	.14
Average	file_complexity	.51	.5	.49	.49	.49	.5	.54	.54	.5	.51	.49	.53	.44	.42	.44	.42	.45	.48		.2	.34	.19	.18	.18
	class_cmplxt	.55	.55	.54	.54	.54	.54	.57	.57	.54	.57	.54	.54	.47	.43	.47	.39	.42	.44	.2		.26	.18	.25	.26
	fn_cmplxt	.56	.57	.55	.57	.57	.57	.55	.57	.56	.57	.55	.55	.5	.49	.51	.45	.52	.46	.34	.26		.15	.19	.19
Organic	new_lines	.19	.19	.19	.19	.19	.19	.19	.18	.2	.19	.19	.2	.17	.17	.19	.15	.16	.13	.19	.18	.15		.13	.13
	new_dup_ln	.18	.18	.18	.18	.18	.18	.18	.18	.17	.16	.17	.18	.14	.14	.18	.14	.16	.14	.18	.25	.19	.13		.01
	new_du_block	.18	.18	.18	.18	.18	.18	.18	.18	.18	.16	.18	.18	.17	.18	.18	.14	.16	.14	.18	.26	.19	.13	.01	

Table 19 Count of NaN from set S_{NaN} indicating missing correlation coefficients (τ_b) for certain pair of metrics

	Cumulative																							Density			Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
ncloc		0	0	0	0	0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
functions	0		0	0	0	0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
statements	0	0		0	0	0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
complexity	0	0	0		0	0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
cmplxt_inClass	0	0	0	0		0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
cmplxt_inFn	0	0	0	0	0		0	0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
classes	0	0	0	0	0	0		0	0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
files	0	0	0	0	0	0	0		0	0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
public_api	0	0	0	0	0	0	0	0		0	0	2	1	1	1	1	1	1	0	1	1	0	1	1
pub_undoc_api	0	0	0	0	0	0	0	0	0		0	2	1	1	1	1	1	1	0	1	1	0	1	1
comment_ln	0	0	0	0	0	0	0	0	0	0		2	1	1	1	1	1	1	0	1	1	0	1	1
directories	2	2	2	2	2	2	2	2	2	2	2		3	3	3	3	3	3	2	3	3	2	3	3
dup_lines	1	1	1	1	1	1	1	1	1	1	1	3		1	1	2	2	2	1	2	2	1	2	2
dup_blocks	1	1	1	1	1	1	1	1	1	1	1	3	1		1	2	2	2	1	2	2	1	2	2
dup_files	1	1	1	1	1	1	1	1	1	1	1	3	1	1		2	2	2	1	2	2	1	2	2
comment-ln-den	1	1	1	1	1	1	1	1	1	1	1	3	1	1	1		1	2	1	1	1	1	2	2
pub-doc-api-den	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	1		2	1	1	1	1	2	2
dup_ln_dens	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	2	2		1	2	2	1	2	2
file_complexity	0	0	0	0	0	0	0	0	0	0	0	2	1	1	1	1	1	1		1	1	0	1	1
class_cmplxt	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	1	1	2	1		1	1	2	2
fn_cmplxt	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	1	1	2	1	1		1	2	2
new_lines	0	0	0	0	0	0	0	0	0	0	0	2	1	1	1	1	1	1	0	1	1		1	1
new_dup_ln	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	1	2	2	1		1
new_du_block	1	1	1	1	1	1	1	1	1	1	1	3	2	2	2	2	2	2	1	2	2	1	1	

Table 20 Count of perfect correlation coefficient (i.e. $\tau_b = 1.0$) from set $S\tau_{b_{vs}}$

	Organic			Average			Density			Cumulative															
	new_du_block	new_dup_ln	new_lines	fn_cmplxt	class_cmplxt	file_complexity	dup_ln_dens	pub-doc-api-den	comment-ln-den	dup_files	dup_blocks	dup_lines	directories	comment_ln	pub_undoc_api	public_api	files	classes	cmplxt_inFn	cmplxt_inClass	complexity	statements	functions	ncloc	
ncloc	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Cumulative
functions	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	
statements	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	
complexity	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11	21		0	0	0	
cmplxt_inClass	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	11		21	0	0	0	
cmplxt_inFn	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		11	11	0	0	0	
classes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	
files	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	
public_api	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		0	0	0	0	0	0	0	0	
pub_undoc_api	0	0	0	0	0	0	0	0	0	0	0	0	0	0		1	0	0	0	0	0	0	0	0	
comment_ln	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	
directories	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	
dup_lines	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	
dup_blocks	0	0	0	0	0	0	0	0	0	1		0	0	0	0	0	0	0	0	0	0	0	0	0	
dup_files	0	0	0	0	0	0	0	0	0		1	0	0	0	0	0	0	0	0	0	0	0	0	0	
comment-ln-den	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Density
pub-doc-api-den	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
dup_ln_dens	0	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
file_complexity	0	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Average
class_cmplxt	0	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
fn_cmplxt	0	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
new_lines	0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Organic
new_dup_ln	1		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
new_du_block		1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Table 21 Mean values for significant (τ_b) from the set $S\tau_{b_{sig}}$. Three gray-scale cell colors indicate three levels of τ_b. Cells with red text indicate weak τ_b

	Cumulative																					Density			Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
ncloc		.97	.98	.97	.97	.97	.93	.93	.94	.91	.86	.79	.66	.67	.67	-.02	-.06	.25	.35	.34	.26	-.02	.05	.05
functions	.97		.96	.98	.98	.97	.93	.92	.94	.92	.86	.79	.67	.68	.68	-.02	-.07	.26	.35	.35	.23	-.02	.05	.05
statements	.98	.96		.97	.97	.97	.92	.91	.94	.91	.86	.78	.66	.68	.68	-.02	-.06	.26	.36	.39	.27	-.02	.05	.05
complexity	.97	.98	.97		1.	1.	.92	.91	.93	.91	.86	.79	.67	.68	.68	-.01	-.06	.26	.36	.37	.26	-.02	.05	.05
cmplxt_inClass	.97	.98	.97	1.		1.	.92	.91	.93	.91	.86	.79	.67	.68	.68	-.01	-.06	.26	.36	.37	.26	-.02	.05	.05
cmplxt_inFn	.97	.98	.97	1.	1.		.92	.91	.94	.91	.86	.79	.67	.68	.68	-.01	-.06	.26	.36	.37	.26	-.02	.05	.05
classes	.93	.93	.92	.92	.92	.92		.98	.92	.9	.84	.81	.69	.7	.71	-.01	-.06	.26	.33	.28	.23	-.02	.06	.06
files	.93	.92	.91	.91	.91	.91	.98		.91	.89	.84	.8	.69	.69	.7	-.01	-.04	.26	.31	.26	.23	-.02	.07	.07
public_api	.94	.94	.94	.93	.93	.94	.92	.91		.95	.85	.78	.67	.71	.71	-.03	-.09	.25	.37	.36	.26	-.02	.05	.05
pub_undoc_api	.91	.92	.91	.91	.91	.91	.9	.89	.95		.81	.76	.69	.72	.69	-.12	-.18	.3	.35	.34	.24	-.04	.01	.01
comment_ln	.86	.86	.86	.86	.86	.86	.84	.84	.85	.81		.78	.61	.61	.69	-.14	-.06	.23	.33	.35	.28	-.03	.04	.04
directories	.79	.79	.78	.79	.79	.79	.81	.8	.78	.76	.78		.66	.69	.73	-.11	-.03	.31	.24	.22	.14	-.04	.04	.04
dup_lines	.66	.67	.66	.67	.67	.67	.7	.69	.71	.69	.61	.66		.9	.88	-.11	-.03	.62	.28	.23	.17	-.04	.15	.11
dup_blocks	.67	.68	.68	.68	.68	.68	.7	.69	.71	.72	.61	.69	.9		.89	-.06	-.05	.6	.31	.24	.12	-.05	.16	.12
dup_files	.67	.68	.68	.68	.68	.68	.71	.7	.71	.72	.69	.73	.88	.89		-.06	.01	.55	.29	.26	.14	-.05	.12	.12
comment-ln-den	-.02	-.02	-.02	-.01	-.01	-.01	-.01	-.01	-.03	-.12	.14	.11	-.11	-.06	-.06		.4	-.11	-.03	.05	-.02	-.07	.01	.01
pub-doc-api-den	-.06	-.07	-.06	-.06	-.06	-.06	-.06	-.04	-.09	-.18	-.06	.03	.03	.05	.01	.4		-.01	-.04	.05	-.01	-.07	-.09	-.09
dup_ln_dens	.25	.26	.26	.26	.26	.26	.26	.25	.3	.23	.31	.62	.6	.55	-.11	-.01		.23	.1	.02	-.08	.11	.11	
file_complexity	.35	.35	.36	.36	.36	.36	.33	.31	.37	.35	.24	.28	.31	.29	-.03	-.04	.23		.78	.53	.54	-.05	.05	.05
class_cmplxt	.34	.35	.39	.37	.37	.37	.28	.26	.36	.34	.22	.23	.24	.26	.05	.05	.1	.78		.78	.53	-.04	-.01	-.01
fn_cmplxt	.26	.23	.27	.26	.26	.26	.23	.23	.26	.24	.14	.17	.12	.14	-.02	-.01	.02	.53	.54		.54	.03	.09	.09
new_lines	-.02	-.02	-.02	-.02	-.02	-.02	-.02	-.02	-.02	-.04	-.03	-.04	-.04	-.05	-.05	-.07	-.07	-.08	-.05	-.04	.03		.37	.37
new_dup_ln	.05	.05	.05	.05	.05	.05	.06	.07	.05	.01	.04	.04	.15	.16	.12	.01	-.09	.11	.05	-.01	.09	.37		.98
new_du_block	.05	.05	.05	.05	.05	.05	.06	.07	.05	.01	.04	.04	.11	.12	.12	.01	-.09	.11	.05	-.01	.09	.37	.98	
Total 'Mean of all significant τ_b'	13.4	13.5	13.4	13.5	13.5	13.5	13.2	13.1	13.3	12.8	12.5	11.8	11.2	11.5	11.5	0.0	-0.6	5.3	6.5	6.2	4.6	0.0	2.5	2.4

Table 22 Sum of significant correlation coefficient (τ_b) from the set $S\tau_{b_{sig}}$. Cells with negative figures are presented with red text

	Cumulative																				Density		Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block
ncloc		20.27	20.48	20.32	20.32	20.32	19.57	19.45	19.65	18.21	18.13	14.95	13.22	13.46	13.45	-.26	-1.15	4.57	6.59	5.45	4.72	-.28	.34	.34
functions	20.27		20.21	20.59	20.59	20.6	19.45	19.32	19.82	18.4	18.1	14.97	13.4	13.62	13.61	-.32	-1.23	4.76	7.06	5.61	4.21	-.27	.34	.34
statements	20.48	20.21		20.42	20.42	20.41	19.27	19.18	19.7	18.19	17.99	14.78	13.29	13.58	13.51	-.36	-1.06	4.66	7.13	5.8	4.86	-.28	.33	.33
complexity	20.32	20.59	20.42		21.	20.98	19.3	19.2	19.63	18.22	18.13	14.96	13.32	13.56	13.55	-.15	-1.12	4.65	7.2	5.94	4.48	-.28	.32	.32
cmplxt_inClass	20.32	20.59	20.42	21.		20.98	19.3	19.2	19.63	18.22	18.13	14.96	13.32	13.56	13.55	-.15	-1.12	4.65	7.2	5.94	4.48	-.28	.32	.32
cmplxt_inFn	20.32	20.6	20.41	20.98	20.98		19.3	19.19	19.64	18.23	18.12	14.96	13.34	13.57	13.56	-.15	-1.13	4.67	7.21	5.93	4.48	-.28	.33	.33
classes	19.57	19.45	19.27	19.3	19.3	19.3		20.5	19.33	18.08	17.73	15.32	13.15	13.31	13.42	-.19	-1.14	4.69	7.21	5.93	4.32	-.24	.4	.4
files	19.45	19.32	19.18	19.2	19.2	19.19	20.5		19.18	17.81	17.7	15.29	13.03	13.2	13.31	-.18	-.7	4.59	5.54	4.46	4.09	-.31	.43	.43
public_api	19.65	19.82	19.7	19.63	19.63	19.64	19.33	19.18		19.07	17.77	14.75	13.39	13.5	13.49	-.56	-1.61	4.66	6.94	5.72	4.62	-.26	.34	.34
pub_undoc_api	18.21	18.4	18.19	18.22	18.22	18.23	18.08	17.81	19.07		16.14	13.77	14.83	13.08	12.91	12.98	-1.98	-3.27	5.11	6.36	5.03	4.04	-.42	.05	.06
comment_ln	18.13	18.1	17.99	18.13	18.13	18.12	17.73	17.7	17.77	16.14		14.83	11.59	11.66	12.46	2.33	-1.18	4.18	6.66	5.66	5.02	-.33	.3	.3
directories	14.95	14.97	14.78	14.96	14.96	14.96	15.32	15.29	14.75	13.77	14.83		11.91	12.43	13.09	1.61	.55	5.23	3.85	3.26	2.39	-.44	.3	.3
dup_lines	13.22	13.4	13.29	13.32	13.32	13.34	13.15	13.03	13.39	11.59	11.59	11.91		18.04	17.66	-1.83	.42	11.69	5.66	3.85	2.83	-.44	1.03	.86
dup_blocks	13.46	13.62	13.58	13.56	13.56	13.57	13.31	13.2	13.5	11.66	12.43	18.04			17.87	-.99	.71	11.41	5.88	4.36	2.18	-.54	1.14	.97
dup_files	13.45	13.61	13.51	13.55	13.55	13.56	13.42	13.31	13.49	12.98	12.46	13.09	17.66	17.87		-.89	.19	10.95	5.54	3.91	2.54	-.47	.95	.96
comment-ln-den	-.26	-.32	-.36	-.15	-.15	-.15	-.19	-.18	-.56	-1.98	2.33	1.61	-1.83	-.99	-.89		7.68	-1.7	-.43	.96	-.27	-.8	.04	.04
pub-doc-api-den	-1.15	-1.23	-1.06	-1.12	-1.12	-1.13	-1.14	-.7	-1.61	-3.27	-1.18	.55	.42	.71	.19	7.68		-.15	-.7	.77	-.21	-.97	-.36	-.37
dup_ln_dens	4.57	4.76	4.66	4.65	4.65	4.67	4.69	4.59	4.66	5.11	4.18	5.23	11.69	11.41	10.95	-1.7	-.15		4.14	1.85	.26	-.7	.98	.97
file_complexity	6.59	7.06	7.13	7.2	7.2	7.21	5.69	5.54	6.94	6.36	6.66	3.85	5.66	5.88	5.54	-.43	-.7	4.14		14.87	9.08	-.46	.3	.3
class_cmplxt	5.45	5.61	5.8	5.94	5.94	5.93	4.53	4.46	5.72	5.03	5.66	3.26	3.85	4.36	3.91	.96	.77	1.85	14.87		10.3	-.33	-.03	-.03
fn_cmplxt	4.72	4.21	4.86	4.48	4.48	4.48	4.32	4.09	4.62	4.04	5.02	2.39	2.83	2.18	2.54	-.27	-.21	.26	9.08	10.3		.3	.28	.28
new_lines	-.28	-.27	-.28	-.28	-.28	-.28	-.24	-.31	-.26	-.42	-.33	-.44	-.44	-.54	-.47	-.8	-.97	-.7	-.46	-.33	.3		7.44	7.38
new_dup_ln	.34	.34	.33	.32	.32	.33	.4	.43	.34	.05	.3	.3	1.03	1.14	.95	.04	-.36	.98	.3	-.03	.28	7.44		19.65
new_du_block	.34	.34	.33	.32	.32	.33	.4	.43	.34	.06	.3	.3	.86	.97	.96	.04	-.37	.97	.3	-.03	.28	7.38	19.65	
Total 'Sum of all significant τ_b'	272	273	273	275	275	275	265	264	269	248	251	218	216	219	219	1	-7	96	122	104	79	7	35	35

Table 23 Count of very strong correlation coefficients ($0.9 \leqslant abs(\tau_b) \leqslant 1.0$) from the set $S\tau_{b_{vs}}$

	Cumulative																									Density			Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total 'Count of Very Strong τ_b'
ncloc		19	19	19	19	19	15	16	18	15	16	8	2	3	5	0	0	0	2	2	1	0	0	0	198
functions	19		19	21	21	21	17	17	19	18	16	9	1	3	4	1	0	0	2	3	1	0	0	0	209
statements	19	19		19	21	21	17	18	19	18	16	9	2	4	5	0	0	0	2	2	1	0	0	0	198
complexity	19	21	19		19	21	16	17	18	18	15	8	2	3	5	1	0	0	2	2	1	0	0	0	206
cmplxt_inClass	19	21	19	19		21	16	17	18	18	15	8	2	3	5	1	0	0	2	2	1	0	0	0	206
cmplxt_inFn	19	21	19	21	21		16	16	19	15	15	8	2	3	5	1	0	0	2	2	1	0	0	0	206
classes	15	17	15	16	16	16		20	15	14	14	8	2	3	5	0	0	0	1	2	0	0	0	0	183
files	16	17	15	17	17	16	20		13	13	14	8	2	3	5	0	0	0	1	3	0	0	0	0	184
public_api	18	19	18	18	18	19	15	13		17	17	9	1	3	5	0	0	0	1	2	0	0	0	0	193
pub_undoc_api	15	15	14	15	15	15	14	13	17		12	8	2	3	3	0	0	0	1	2	0	0	0	0	164
comment_ln	16	16	16	15	15	15	14	14	17	12		9	1	2	3	0	0	0	1	2	0	0	0	0	168
directories	8	9	9	8	8	8	12	12	9	8	9		2	3	4	0	0	0	0	1	0	0	0	0	110
dup_lines	2	1	2	2	2	2	2	2	1	2	1	2		12	13	0	0	1	0	0	0	0	0	0	47
dup_blocks	3	3	4	3	3	3	3	3	3	3	2	3	12		14	0	0	0	0	0	0	0	0	0	63
dup_files	5	4	5	5	5	5	5	5	5	3	3	4	13	14		0	0	0	0	0	0	0	0	0	81
comment-ln-den	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0		0	0	0	0	1	0	0	0	5
pub-doc-api-den	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0	0	0
dup_ln_dens	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0		0	0	0	0	0	0	2
file_complexity	2	2	2	2	2	2	1	1	1	1	1	1	0	0	0	0	0	0		8	0	0	0	0	25
class_cmplxt	2	3	2	2	2	2	2	3	2	2	2	2	1	0	0	0	0	0	8		0	0	0	0	33
fn_cmplxt	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0		0	0	0	7
new_lines	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0
new_dup_ln	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		20	20
new_du_block	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20		20

Table 24 Count of strong correlation coefficients ($0.7 \leqslant abs(\tau_b) < 0.9$) from the set $S\tau_{b_s}$

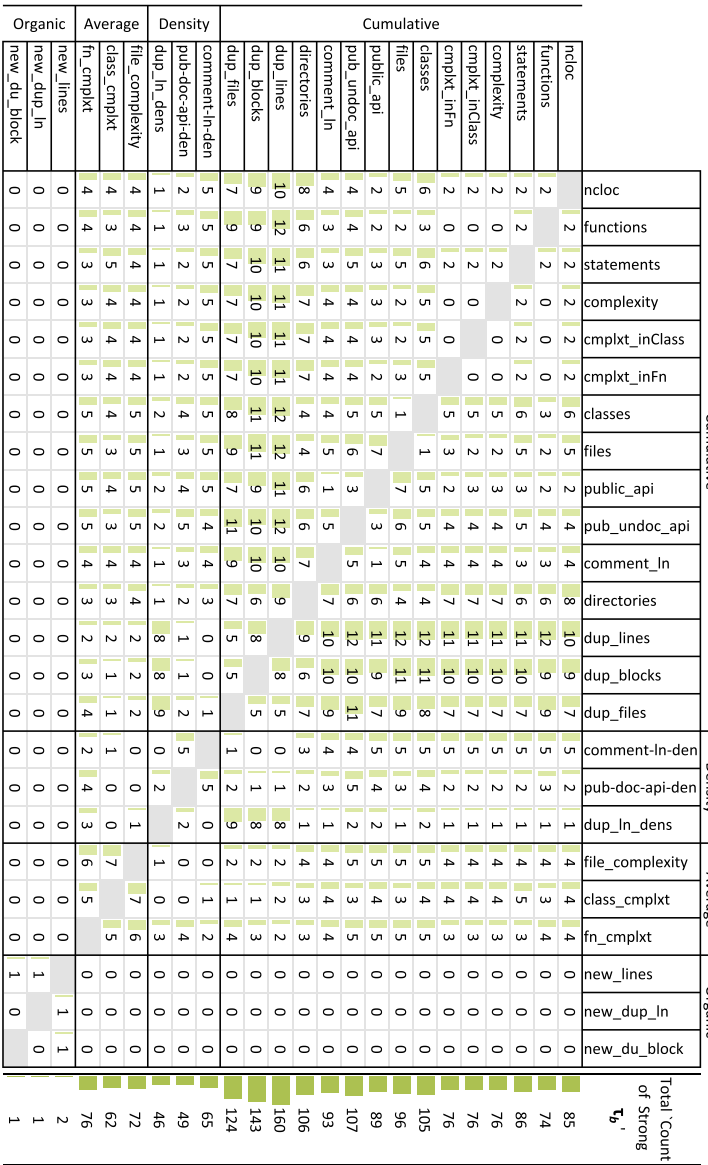


Table 25 Count of moderate correlation coefficients ($0.4 \leqslant abs(\tau_b) < 0.7$) from the set $S\tau_{b_m}$

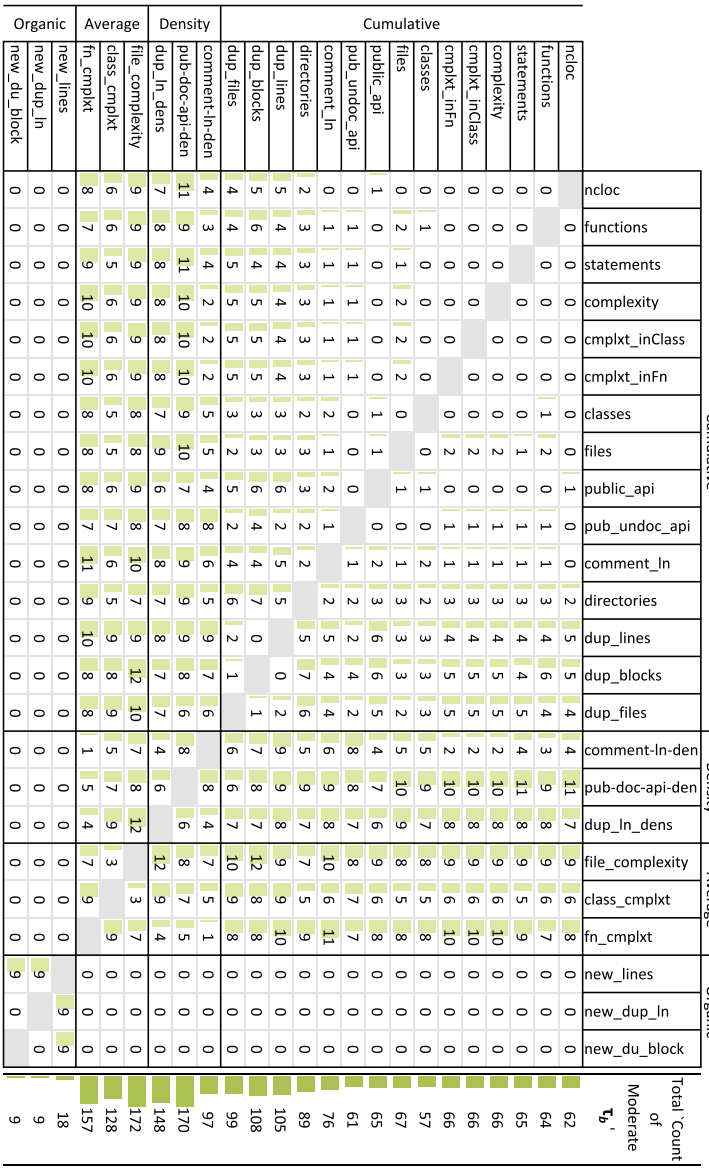


Table 26 Count of weak correlation coefficients ($0 \leqslant abs(\tau_b) < 0.4$) from the set $S\tau_{b_w}$

		Cumulative															Density			Average			Organic			
		ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total `Count of Weak τ_b'
Cumulative	ncloc		0	0	0	0	0	0	0	0	1	1	1	3	3	4	8	5	10	4	4	5	12	7	7	75
	functions	0		0	0	0	0	0	0	0	0	1	1	3	2	3	8	6	9	5	4	6	12	7	7	74
	statements	0	0		0	0	0	0	0	0	0	1	1	3	2	3	7	5	9	5	3	5	12	7	7	70
	complexity	0	0	0		0	0	0	0	0	0	1	1	3	2	3	8	6	9	5	4	3	12	7	7	71
	cmplxt_inClass	0	0	0	0		0	0	0	0	0	1	1	3	2	3	8	6	9	5	4	3	12	7	7	71
	cmplxt_inFn	0	0	0	0	0		0	0	0	0	1	1	3	2	3	8	6	9	5	4	3	12	7	7	71
	classes	0	0	0	0	0	0		0	0	1	1	1	2	2	3	7	5	9	3	5	6	12	7	7	71
	files	0	0	0	0	0	0	0		0	1	1	0	2	2	3	7	6	8	4	6	5	13	6	6	70
	public_api	0	0	0	0	0	0	0	0		0	1	1	2	1	2	8	6	11	4	4	5	12	7	7	71
	pub_undoc_api	1	0	0	0	0	0	1	1	0		2	2	3	1	2	5	5	8	4	3	5	11	6	6	66
	comment_ln	1	1	1	1	1	1	1	1	1	2		1	3	3	2	7	7	9	5	4	3	13	7	7	82
	directories	1	1	1	1	1	1	1	0	1	2	1		2	2	1	7	6	9	5	6	5	12	7	7	80
	dup_lines	3	3	3	3	3	3	2	2	2	3	3	2		0	0	7	4	2	9	6	5	11	7	8	91
	dup_blocks	3	2	2	2	2	2	2	2	1	1	3	2	0		0	10	4	3	5	9	7	10	7	8	87
	dup_files	4	3	3	3	3	3	3	3	2	2	2	1	0	0		8	6	4	7	5	6	10	8	8	94
Density	comment-ln-den	8	8	7	8	8	8	7	7	8	5	7	7	7	10	8		6	12	10	12	11	11	3	3	181
	pub-doc-api-den	5	6	5	6	6	6	5	6	6	5	7	6	4	4	6	6		8	8	9	7	13	4	4	142
	dup_ln_dens	10	9	9	9	9	9	9	8	11	8	9	9	2	3	4	12	8		5	10	9	9	9	9	189
Average	file_complexity	4	5	5	5	5	5	3	4	4	4	5	5	9	5	7	10	8	5		1	4	10	6	6	125
	class_cmplxt	4	4	3	4	4	4	5	6	4	3	4	6	6	9	5	12	9	10	1		5	9	4	4	125
	fn_cmplxt	5	6	5	3	3	3	6	5	5	5	3	5	5	7	6	11	7	9	4	5		9	3	3	123
Organic	new_lines	12	12	12	12	12	12	12	13	12	11	13	12	11	10	10	11	13	9	10	9	9		10	10	257
	new_dup_ln	7	7	7	7	7	7	7	6	7	6	7	7	7	7	8	3	4	9	6	4	3	10		0	143
	new_du_block	7	7	7	7	7	7	7	6	7	6	7	7	8	8	8	3	4	9	6	4	3	10	0		145

Table 27 Sum of very strong correlation coefficients ($0.9 \leqslant abs(\tau_b) \leqslant 1.0$) from the set $S\tau_{b_{vs}}$. Cells with negative figures are presented with red text color instead of horizontal data bar. Dot (.) in a cell indicates zero value

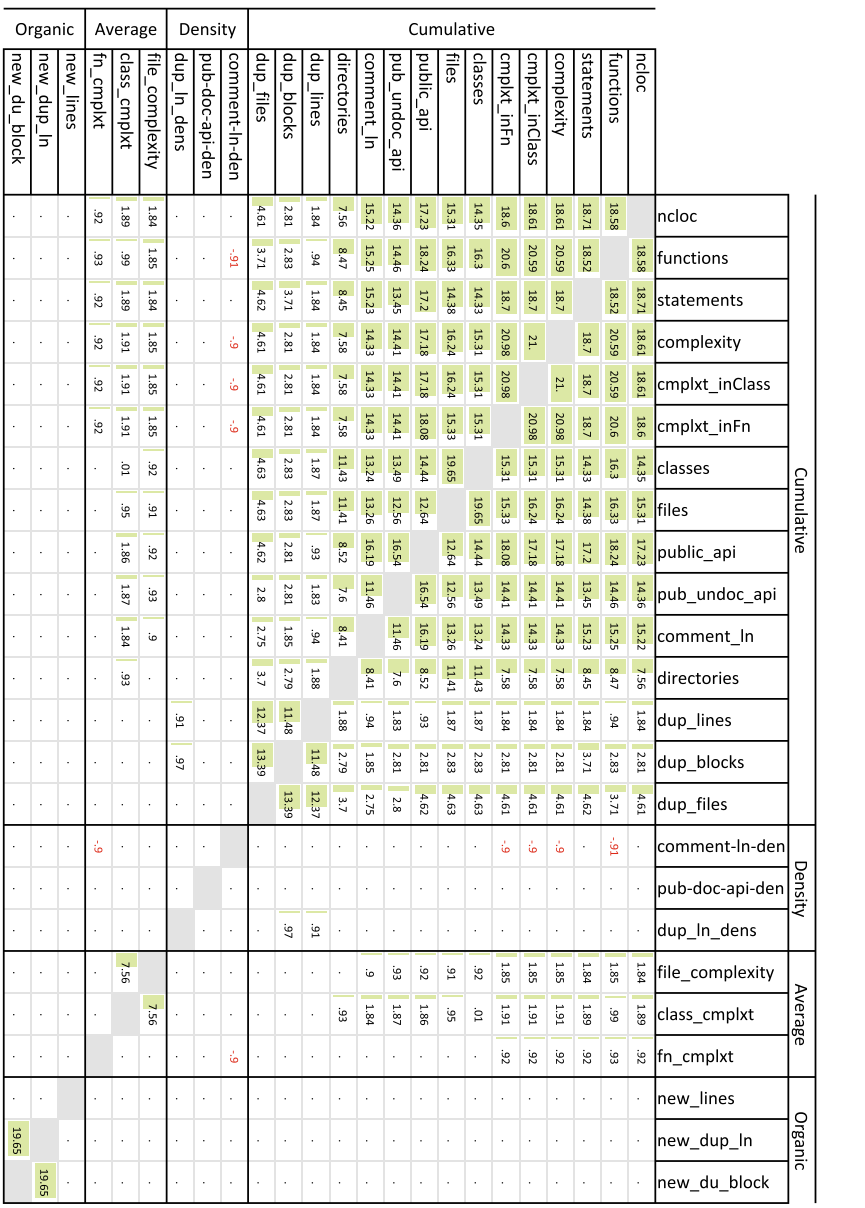


Table 28 Sum of strong correlation coefficients ($0.7 \leqslant abs(\tau_b) < 0.9$) from the set $S\tau_{b_s}$. Cells with negative figures are presented with red text color instead of horizontal data bar. Dot (.) in a cell indicates zero value

	Cumulative																					Density		Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total Sum of Strong τ_b
ncloc		1.69	1.78	1.71	1.71	1.71	5.21	4.14	1.79	3.48	3.22	6.59	7.97	7.01	5.58	-.83	-1.56	.74	1.44	1.53	1.67	.	.	.	57
functions	1.69		1.69	.	.	.	2.46	1.74	1.58	3.42	2.51	5.02	9.58	6.99	7.15	.79	-2.3	.74	1.44	2.41	1.67	.	.	.	49
statements	1.78	1.69		1.71	1.71	.	4.93	4.14	2.5	4.29	2.39	4.9	8.65	7.48	5.52	-.84	-1.47	.74	1.46	2.26	2.37	.	.	.	58
complexity	1.71	.	1.71		.	.	3.99	3.99	2.45	3.32	3.44	5.92	8.64	7.68	5.52	.81	-1.59	.74	1.46	1.56	2.38	.	.	.	51
cmplxt_inClass	1.71	.	1.71	.		.	3.99	3.99	1.69	3.32	3.44	5.92	8.64	7.68	5.52	.81	-1.59	.74	1.46	1.56	2.38	.	.	.	51
cmplxt_inFn	1.71	.	1.71	.	.		4.	2.59	1.55	3.33	3.44	5.92	8.65	7.68	5.52	.81	-1.59	.74	1.46	1.56	2.38	.	.	.	51
classes	5.21	2.46	4.93	3.99	3.99	4.		.85	4.39	4.31	3.47	3.14	9.43	8.52	6.42	-.8	-3.14	1.52	2.27	3.22	2.51	.	.	.	71
files	4.14	1.74	4.14	1.69	1.69	2.59	.85		6.08	5.01	4.22	3.09	9.35	8.46	7.06	-.8	-2.38	.75	2.28	2.33	2.55	.	.	.	65
public_api	1.79	1.58	2.5	2.45	2.45	1.55	4.39	6.08		2.53	.73	4.92	8.85	6.91	5.58	-.77	-1.6	1.45	2.31	1.59	2.55	.	.	.	58
pub_undoc_api	3.48	3.42	4.29	3.32	3.32	3.33	4.31	5.01	2.53		4.18	5.03	9.51	7.68	8.82	-1.5	-3.97	1.53	2.39	.82	2.54	.	.	.	70
comment_ln	3.22	2.51	2.39	3.44	3.44	3.44	3.47	4.22	.73	4.18		5.68	8.	7.69	7.25	1.75	-.76	.7	1.63	1.61	3.24	.	.	.	68
directories	6.59	5.02	4.9	5.92	5.92	5.92	3.14	3.09	4.92	5.03	5.68		7.01	4.64	5.71	.96	-1.58	.76	.19	.75	.82	.	.	.	75
dup_lines	7.97	9.58	8.65	8.64	8.64	8.65	9.43	9.35	8.85	9.51	8.	7.01		6.57	4.24	.	-.84	6.37	1.52	.05	-.06	.	.	.	122
dup_blocks	7.01	6.99	7.48	7.68	7.68	7.68	8.52	8.46	6.91	7.68	7.69	4.64	6.57		3.96	.	-.82	6.53	1.57	.82	.69	.	.	.	108
dup_files	5.58	7.15	5.52	5.52	5.52	5.52	6.42	7.06	5.58	8.82	7.25	5.71	4.24	3.96		-.76	-1.57	7.18	1.57	.82	1.38	.	.	.	92
comment-ln-den	-.83	.79	-.84	.81	.81	.81	-.8	-.8	-.77	-1.5	1.75	.96	.	.	-.76		3.95	.	.	.8	.08	.	.	.	4
pub-doc-api-den	-1.56	-2.3	-1.47	-1.59	-1.59	-1.59	-3.14	-2.38	-1.6	-3.97	-.76	-1.58	-.84	-.82	-1.57	3.95		-.15	.	.	.13	.	.	.	-23
dup_ln_dens	.74	.74	.74	.74	.74	.74	1.52	.75	1.45	1.53	.7	.76	6.37	6.53	7.18	.	-.15		.75	.	-.67	.	.	.	31
file_complexity	1.44	1.44	1.46	1.46	1.46	1.46	2.27	2.28	2.31	2.39	1.63	.19	1.52	1.57	1.57	.	.	.75		5.44	4.66	.	.	.	35
class_cmplxt	1.53	2.41	2.26	1.56	1.56	1.56	3.22	2.33	1.59	.82	1.61	.75	.05	.82	.82	.8	.	.	5.44		4.15	.	.	.	33
fn_cmplxt	1.67	2.37	2.38	2.38	2.38	2.38	2.51	2.55	2.55	2.54	3.24	.82	-.06	.69	1.38	.08	.13	-.67	4.66	4.15		.	.	.	37
new_lines	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.75	.73	1
new_dup_ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.75		.	1
new_du_block	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.73	.		1

Table 29 Sum of moderate correlation coefficients ($0.4 \leqslant abs(\tau_b) < 0.7$) from the set $S\tau_{b_m}$. Cells with negative figures are presented with red text color instead of horizontal data bar. Dot (.) in a cell indicates zero value

	Cumulative																									Density		Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total `Sum of Moderate τ_b'
ncloc	.	.	.	.	.	.	.	.	.63	.	1.09	3.13	3.33	2.59	-.76	.48	1.77	2.76	2.06	1.1	.	.	.	.	18
functions	.	.	.	.	.	.	.7	1.25	.	.51	.66	1.77	2.45	3.83	2.39	-1.4	1.54	2.19	2.72	2.03	.64	.	.	.	21
statements	.	.	.	.	.	.	.	1.27	.66	.46	.7	1.72	2.43	2.41	3.	-.8	.49	2.16	2.96	1.47	.47	.	.	.	18
complexity	.	.	.	.	.	.	.	1.27	.	.49	.68	1.74	2.44	3.11	3.06	-.96	.92	2.15	2.87	2.06	.84	.	.	.	21
cmplxt_inClass	.	.	.	.	.	.	.	1.27	.	.49	.68	1.74	2.44	3.11	3.06	-.96	.92	2.15	2.87	2.06	.84	.	.	.	21
cmplxt_inFn	.	.	.	.	.	.	1.27	.	.	.49	.68	1.74	2.44	3.11	3.06	-.96	.91	2.16	2.87	2.06	.83	.	.	.	21
classes	.	.7	.	.	.	.	.	.	.5	.	1.37	1.16	1.76	1.91	1.93	-.48	1.86	1.58	1.9	1.4	1.07	.	.	.	17
files	.	1.25	.66	1.27	1.27	1.27	.	.	.46	.	.57	.78	1.76	1.93	1.25	-.55	1.39	1.83	1.87	1.41	1.07	.	.	.	19
public_api	.63	.	.	.	.	.	.5	.46	.	1.14	1.7	3.55	3.99	3.2	-.85	-.42	1.14	2.82	2.	1.06	.	.	.	.	21
pub_undoc_api	.	.51	.46	.49	.49	.49	.	.	.	.	.63	1.21	1.3	2.63	1.17	-1.03	.34	1.54	2.32	2.36	.57	.	.	.	15
comment_ln	.	.66	.7	.68	.68	.68	1.37	.57	1.14	.63	.	1.11	2.96	2.54	2.48	-.2	.37	2.02	3.5	1.97	1.4	.	.	.	25
directories	1.09	1.77	1.72	1.74	1.74	1.74	1.16	.78	1.7	1.21	1.11	.	2.81	4.29	3.39	.3	1.63	3.01	2.87	1.47	.52	.	.	.	36
dup_lines	3.13	2.45	2.43	2.44	2.44	2.44	1.76	1.76	3.55	1.3	2.96	2.81	.	1.05	-2.97	.67	4.61	2.95	4.12	2.43	.	.	.	.	42
dup_blocks	3.33	3.83	2.41	3.11	3.11	3.11	1.91	1.93	3.99	2.63	2.54	4.29	.	.	-1.44	1.11	2.77	3.4	3.57	1.13	.	.	.	.	47
dup_files	2.59	2.39	3.	3.06	3.06	3.06	1.93	1.25	3.2	1.17	2.48	3.39	1.05	.53	.	1.12	3.46	2.52	3.02	.85	.	.	.	.	42
comment-ln-den	-.76	-1.4	-.8	-.96	-.96	-.96	-.48	-.55	-.85	-1.03	-.2	.3	-.97	-1.44	-.88	.	2.45	-2.13	.15	-.63	.45	.	.	.	-14
pub-doc-api-den	.48	1.54	.49	.92	.92	.91	1.86	1.39	-.42	.34	-.37	1.63	.67	1.11	1.12	2.45	.	-.14	.65	.99	-.81	.	.	.	16
dup_ln_dens	1.77	2.19	2.16	2.15	2.15	2.16	1.58	1.83	1.14	1.54	2.02	3.01	4.61	2.77	3.46	-2.13	-.14	.	2.53	1.69	.94	.	.	.	37
file_complexity	2.76	2.72	2.96	2.87	2.87	2.87	1.9	1.87	2.82	2.32	3.5	2.87	2.95	3.4	2.52	.15	.65	2.53	.	1.64	4.35	.	.	.	51
class_cmplxt	2.06	2.03	1.47	2.06	2.06	2.06	1.4	1.41	2.	2.36	1.97	1.47	4.12	3.57	3.02	-.63	.99	1.69	1.64	.	5.15	.	.	.	42
fn_cmplxt	1.1	.64	.47	.84	.84	.83	1.07	1.07	1.06	.57	1.4	.52	2.43	1.13	.85	.45	-.81	.94	4.35	5.15	.	.	.	.	25
new_lines	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3.99	3.96	8
new_dup_ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3.99	.	3.96	4
new_du_block	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3.96	3.96	.	4

Table 30 Sum of weak correlation coefficients ($0 \leqslant abs(\tau_b) < 0.4$) from the set $S\tau_{b_w}$. Cells with negative figures are presented with red text color. Dot (.) in a cell indicates zero value

	Cumulative																			Density			Average			Organic		
	ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens	file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	Total Sum of Weak $\tau_{b'}$
ncloc		.	.	.	.	.	.	.	.	.37	-.32	-.28	.28	.31	.67	1.33	-.07	2.05	.55	-.03	1.04	-.28	.34	.34	6
functions	.		.	.	.	.	.	.	.	-.33	-.29	.43	-.03	.36	1.2	-.47	1.83	1.05	.18	.97	-.27	.34	.34	5
statements	.	.		.	.	.	.	.	.	-.33	-.28	.37	-.03	.36	1.28	-.08	1.75	.88	.17	1.1	-.28	.33	.33	6
complexity	.	.	.		.	.	.	.	.	-.33	-.28	.4	-.03	.37	.9	-.46	1.76	1.02	.41	.35	-.28	.32	.32	4
cmplxt_inClass	.	.	.	.		.	.	.	.	-.33	-.28	.4	-.03	.37	.9	-.46	1.76	1.02	.41	.35	-.28	.32	.32	4
cmplxt_inFn	.	.	.	.	.		.	.	.	-.33	-.28	.4	-.03	.37	.9	-.46	1.76	1.03	.4	.35	-.28	.33	.33	4
classes	.	.	.	.	.	.		.	.	.28	-.4	.09	.05	.44	1.09	.14	1.59	.6	-.11	.73	-.24	.4	.4	5
files	.	.	.	.	.	.	.		.	.24	-.36	.05	-.02	.37	1.17	.28	2.01	.47	-.23	.47	-.31	.43	.43	5
public_api	.	.	.	.	.	.	.	.		.	-.3	.06	-.21	.09	1.06	.41	2.08	.88	.27	1.	-.26	.34	.34	5
pub_undoc_api	.37	.	.	.	.	.	.28	.24	.		-.13	-.06	.45	-.2	.19	.56	.35	2.05	.72	-.01	.93	-.42	.05	.06	-1
comment_ln	-.32	-.33	-.33	-.33	-.33	-.33	-.36	-.36	-.3	-.13		-.38	-.31	-.43	-.03	.78	-.05	1.46	.63	.24	.38	-.33	.3	.3	3
directories	-.28	-.29	-.28	-.28	-.28	-.28	-.4	.	-.4	-.06	-.38		.21	.72	.29	.34	.51	1.46	.79	.11	1.04	-.44	.3	.3	7
dup_lines	.28	.43	.37	.4	.4	.4	.09	.05	.06	.45	-.31	.21		.	.	1.14	.59	-.2	1.19	-.32	.46	-.44	1.03	.86	5
dup_blocks	.31	-.03	-.03	-.03	-.03	-.03	.05	-.02	-.21	-.2	-.43	.72	.		.	.45	.42	1.14	.91	-.03	.37	-.54	1.14	.97	9
dup_files	.67	.36	.36	.37	.37	.37	.44	.37	.09	.19	-.03	.29	.	.		.74	.65	.31	1.45	.06	.32	-.47	.95	.96	15
comment-ln-den	1.33	1.2	1.28	.9	.9	.9	1.09	1.17	1.06	.56	.78	.34	1.14	.45	.74		1.28	.43	-.57	.78	.11	-.8	.04	.04	0
pub-doc-api-den	-.07	-.47	-.08	-.46	-.46	-.46	.14	.28	.41	.35	-.05	.51	.59	.42	.65	1.28		.15	-1.34	-.23	.46	-.97	-.36	-.37	26
dup_ln_dens	2.05	1.83	1.75	1.76	1.76	1.76	1.59	2.01	2.08	2.05	1.46	1.46	-.2	1.14	.31	.43	.15		.86	.16	.	-.7	.98	.97	13
file_complexity	.55	1.05	.88	1.02	1.02	1.03	.6	.47	.88	.72	.63	.79	1.19	.91	1.45	-.57	-1.34	.86		.24	.07	-.46	.3	.3	3
class_cmplxt	-.03	.18	.17	.41	.41	.4	-.11	-.23	.27	-.01	.24	.11	-.32	-.03	.06	.78	-.23	.16	.24		1.01	-.33	-.03	-.03	3
fn_cmplxt	1.04	.97	1.1	.35	.35	.35	.73	.47	1.	.93	.38	1.04	.46	.37	.32	.11	.46	.	.07	1.01		.3	.28	.28	12
new_lines	-.28	-.27	-.28	-.28	-.28	-.28	-.24	-.31	-.26	-.42	-.33	-.44	-.44	-.54	-.47	-.8	-.97	-.7	-.46	-.33	.3		2.69	2.69	-3
new_dup_ln	.34	.34	.33	.32	.32	.32	.4	.43	.34	.05	.3	.3	1.03	1.14	.95	.04	-.36	.98	.3	-.03	.28	2.69		.	11
new_du_block	.34	.34	.33	.32	.32	.33	.4	.43	.34	.06	.3	.3	.86	.97	.96	.04	-.37	.97	.86	-.03	.28	2.69	.		10

Table 31 Standard deviations, count of non-significant τ_b, and count of perfect τ_b of significant correlation coefficients for correlations between organic$_t$ and metrics from cumulative and density categories from the set $S\tau_{b_{all}}$. Dot (.) in a cell indicates zero value

			Cumulative															Density		
			ncloc	functions	statements	complexity	cmplxt_inClass	cmplxt_inFn	classes	files	public_api	pub_undoc_api	comment_ln	directories	dup_lines	dup_blocks	dup_files	comment-ln-den	pub-doc-api-den	dup_ln_dens
Standard Deviation	Organic (Transformed)	_ncloc	.17	.17	.17	.17	.17	.17	.17	.17	.17	.16	.17	.16	.14	.14	.15	.13	.14	.12
		_functions	.16	.16	.16	.16	.16	.16	.16	.15	.15	.14	.16	.14	.14	.15	.14	.06	.12	.11
		_statements	.18	.17	.18	.17	.17	.17	.18	.18	.17	.16	.18	.17	.16	.14	.15	.1	.14	.11
		_complexity	.16	.16	.16	.16	.16	.15	.16	.16	.15	.14	.16	.15	.14	.15	.14	.11	.14	.13
		_cmplxt_inClass	.16	.16	.16	.16	.16	.15	.16	.16	.15	.14	.16	.15	.14	.15	.14	.11	.14	.13
		_cmplxt_inFn	.16	.16	.16	.16	.16	.16	.16	.16	.16	.14	.16	.15	.14	.15	.14	.11	.14	.13
		_classes	.12	.12	.12	.12	.12	.12	.12	.12	.12	.09	.12	.09	.09	.08	.09	.07	.11	.12
		_files	.11	.11	.11	.11	.11	.11	.11	.11	.11	.11	.11	.11	.1	.11	.1	.07	.02	.16
		_public_api	.14	.16	.14	.16	.16	.16	.15	.13	.13	.13	.15	.14	.11	.14	.12	.05	.08	.12
		_pub_undoc_api	.12	.12	.12	.12	.12	.12	.13	.12	.12	.12	.14	.11	.1	.11	.1	.04	.08	.12
		_comment_ln	.14	.14	.13	.14	.14	.14	.15	.16	.15	.14	.14	.13	.15	.12	.15	.12	.13	.13
		_directories	.17	.17	.17	.17	.17	.17				.05	.18	.01			.14	.04	.03	.12
		_dup_lines	.18	.18	.18	.18	.18	.18	.15	.18	.17	.01	.17	.18	.13	.15	.14	.02	.15	.09
		_dup_blocks	.15	.16	.15	.16	.16	.16	.13	.16	.15	.01	.15	.17	.12	.12	.13	.03	.1	.09
		_dup_files	.16	.16	.16	.16	.16	.16	.17	.17	.15		.12	.18	.17	.16	.12	.08	.02	.09
Count of non-significant tb	Organic (Transformed)	_ncloc	11.	11.	11.	11.	11.	11.	11.	11.	12.	12.	12.	11.	10.	12.	12.	16.	13.	12.
		_functions	11.	11.	11.	12.	12.	12.	12.	11.	10.	11.	12.	12.	10.	12.	9.	16.	13.	16.
		_statements	11.	11.	10.	11.	11.	11.	12.	12.	12.	12.	11.	11.	11.	11.	9.	14.	13.	11.
		_complexity	11.	12.	12.	12.	12.	12.	12.	12.	12.	13.	10.	10.	11.	13.	12.	15.	13.	13.
		_cmplxt_inClass	11.	12.	12.	12.	12.	12.	12.	12.	12.	13.	10.	10.	11.	13.	12.	15.	13.	13.
		_cmplxt_inFn	11.	12.	12.	12.	12.	12.	12.	12.	12.	13.	10.	10.	11.	13.	11.	15.	13.	14.
		_classes	14.	14.	14.	14.	14.	14.	13.	13.	13.	12.	13.	13.	17.	16.	15.	15.	16.	17.
		_files	14.	14.	14.	14.	14.	14.	15.	15.	14.	14.	15.	13.	18.	16.	15.	16.	18.	18.
		_public_api	15.	16.	15.	16.	16.	16.	14.	13.	13.	14.	15.	15.	14.	15.	14.	15.	14.	15.
		_pub_undoc_api	13.	13.	13.	13.	13.	13.	13.	13.	14.	14.	15.	13.	15.	15.	14.	15.	14.	17.
		_comment_ln	12.	12.	12.	13.	13.	13.	13.	14.	13.	15.	12.	11.	13.	14.	13.	13.	15.	15.
		_directories	17.	17.	17.	17.	17.	17.	18.	18.	18.	17.	17.	16.	17.	18.	15.	16.	15.	15.
		_dup_lines	18.	18.	18.	18.	18.	18.	17.	18.	18.	18.	18.	16.	15.	16.	16.	17.	17.	14.
		_dup_blocks	18.	18.	18.	18.	18.	18.	17.	18.	18.	18.	18.	16.	15.	15.	16.	15.	15.	14.
		_dup_files	18.	18.	18.	18.	18.	18.	18.	18.	18.	19.	17.	16.	18.	17.	15.	15.	17.	15.
Count of Perfect Correlations	Organic (Transformed)	_ncloc	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_functions	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_statements	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_complexity	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_cmplxt_inClass	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_cmplxt_inFn	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_classes	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_files	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_public_api	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_pub_undoc_api	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_comment_ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_directories	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_dup_lines	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_dup_blocks	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_dup_files	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Table 32 Standard deviations, count of non-significant τ_b, and count of perfect τ_b of significant correlation coefficients for correlations between organic$_t$ and metrics from average, organic and organic$_t$ categories from the set $S\tau_{b_{all}}$. Dot (.) in a cell indicates zero value

			Average			Organic			Organic (Transformed)																
			file_complexity	class_cmplxt	fn_cmplxt	new_lines	new_dup_ln	new_du_block	_ncloc	_functions	_statements	_complexity	_cmplxt_inClass	_cmplxt_inFn	_classes	_files	_public_api	_pub_undoc_api	_comment_ln	_directories	_dup_lines	_dup_blocks	_dup_files
Standard Deviation	Organic (Transformed)	_ncloc	.17	.19	.16	.11	.12	.12		.05	.04	.04	.04	.04	.1	.11	.09	.08	.1	.11	.1	.13	.13
		_functions	.15	.17	.12	.1	.12	.12	.05		.07	.07	.07	.07	.09	.11	.05	.06	.12	.14	.11	.14	.14
		_statements	.16	.16	.15	.11	.12	.12	.04	.07		.03	.03	.03	.1	.11	.08	.09	.09	.09	.11	.14	.14
		_complexity	.15	.14	.13	.09	.13	.13	.04	.07	.03		.	.01	.11	.12	.08	.08	.12	.11	.11	.14	.14
		_cmplxt_inClass	.15	.14	.13	.09	.13	.13	.04	.07	.03	.		.01	.11	.12	.08	.08	.12	.11	.11	.14	.14
		_cmplxt_inFn	.15	.14	.14	.1	.12	.12	.04	.07	.03	.01	.01		.11	.12	.08	.08	.12	.11	.11	.14	.14
		_classes	.03	.09	.15	.09	.1	.1	.1	.09	.1	.11	.11	.11		.08	.11	.12	.13	.16	.14	.15	.17
		_files	.02	.06	.14	.09	.12	.12	.11	.11	.11	.12	.12	.12	.08		.12	.12	.13	.16	.13	.14	.15
		_public_api	.16	.16	.13	.11	.1	.1	.09	.05	.08	.08	.08	.08	.11	.12		.06	.09	.14	.13	.14	.16
		_pub_undoc_api	.16	.13	.14	.11	.09	.09	.08	.06	.09	.08	.08	.08	.12	.12	.06		.11	.14	.13	.14	.16
		_comment_ln	.11	.12	.12	.12	.11	.11	.1	.12	.09	.12	.12	.12	.13	.13	.09	.11		.12	.13	.11	.12
		_directories		.04	.12	.08	.08	.08	.11	.14	.09	.11	.11	.11	.16	.16	.14	.14	.12		.12	.12	.15
		_dup_lines	.12	.24	.24	.05	.14	.14	.1	.11	.11	.11	.11	.11	.14	.13	.13	.13	.13	.12		.08	.16
		_dup_blocks	.12		.22	.07	.14	.14	.13	.14	.14	.14	.14	.14	.15	.14	.14	.14	.11	.12	.08		.14
		_dup_files	.09	.11	.15	.1	.14	.15	.13	.14	.14	.14	.14	.14	.17	.15	.16	.16	.12	.15	.16	.14	
Count of non-significant tb	Organic (Transformed)	_ncloc	14.	15.	13.	.	1.	1.		.	.	.	.	.	.	.	.	.	2.	2.	.	1.	.
		_functions	14.	15.	14.	.	.	.	.		.	.	.	.	.	.	.	.	.	3.	.	.	.
		_statements	13.	15.	12.	.	2.	2.	.	.		.	.	.	.	.	.	.	2.	3.	1.	.	.
		_complexity	13.	14.	13.	.	.	.	.	.	.		.	.	.	.	.	.	.	5.	.	1.	.
		_cmplxt_inClass	13.	14.	13.	.	.	.	.	.	.	.		.	.	.	.	.	.	5.	.	1.	.
		_cmplxt_inFn	13.	14.	13.	.	.	.	.	.	.	.	.		.	.	.	.	.	5.	.	1.	.
		_classes	18.	15.	13.	.	2.	2.	.	.	.	.	.	.		.	.	.	1.	1.	3.	2.	2.
		_files	18.	18.	15.	1.	1.	1.	.	.	.	.	.	.	.		.	.	1.	1.	1.	1.	1.
		_public_api	17.	18.	16.	.	2.	2.	.	.	.	.	.	.	.	.		.	1.	1.	2.	2.	2.
		_pub_undoc_api	17.	17.	17.	.	2.	2.	.	.	.	.	.	.	.	.	.		1.	2.	2.	2.	1.
		_comment_ln	13.	12.	15.	.	3.	3.	2.	.	2.	.	.	.	1.	1.	1.	1.		4.	3.	3.	3.
		_directories	19.	16.	14.	4.	5.	5.	2.	3.	3.	5.	5.	5.	1.	1.	1.	2.	4.		5.	5.	5.
		_dup_lines	17.	17.	17.	4.	.	.	.	.	1.	.	.	.	3.	1.	2.	2.	3.	5.		.	.
		_dup_blocks	17.	18.	17.	3.	.	.	1.	.	.	1.	1.	1.	2.	1.	2.	2.	3.	5.	.		.
		_dup_files	16.	16.	16.	5.	1.	1.	.	.	.	.	.	.	2.	1.	2.	1.	3.	5.	.	.	
Count of Perfect Correlations	Organic (Transformed)	_ncloc	.	.	.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.	.
		_functions	.	.	.	.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.	.
		_statements	.	.	.	.	.	.	.	.		.	.	.	.	.	.	.	.	.	.	.	.
		_complexity	.	.	.	.	.	.	.	.	.		21.	6.	.	.	.	.	.	.	.	.	.
		_cmplxt_inClass	.	.	.	.	.	.	.	.	.	21.		6.	.	.	.	.	.	.	.	.	.
		_cmplxt_inFn	.	.	.	.	.	.	.	.	.	6.	6.		.	.	.	.	.	.	.	.	.
		_classes	.	.	.	.	.	.	.	.	.	.	.	.		.	.	.	.	.	.	.	.
		_files	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.	.	.	.	.	.
		_public_api	.	.	.	.	.	.	.	.	.	.	.	.	.	.		1.	.	.	.	.	.
		_pub_undoc_api	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.		.	.	.	.	.
		_comment_ln	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.	.	.
		_directories	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.	.
		_dup_lines	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		.	.
		_dup_blocks	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.		1.
		_dup_files	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1.	

References

Aggarwal CC (2013) Outlier Analysis. Springer Publishing Company, Incorporated, Berlin

Baxter G, Frean M, Noble J, Rickerby M, Smith H, Visser M, Melton H, Tempero E (2006) Understanding the shape of java software. In: Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, ACM, New York, OOPSLA '06, pp 397–412. https://doi.org/10.1145/1167473.1167507

Chidamber SR, Darcy DP, Kemerer CF (1998) Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Trans Softw Eng 24(8):629–639

Coleman D, Ash D, Lowther B, Oman P (1994) Using metrics to evaluate software system maintainability. Computer 27(8):44–49. https://doi.org/10.1109/2.303623

Concas G, Marchesi M, Pinna S, Serra N (2007) Power-Laws in a large object-Oriented software system. IEEE Trans Softw Eng 33(10):687–708. https://doi.org/10.1109/TSE.2007.1019

Crawford L, Hobbs JB, Turner JR (2002) Investigation of potential classification systems for projects. In: Proceedings of the 2nd PMI Research Conference, Project Management Institute, Seattle, Washington, pp 181–190

Croux C, Dehon C (2010) Influence functions of the spearman and kendall correlation measures. Statistical Methods &, Applications 19(4):497–515. https://doi.org/10.1007/s10260-010-0142-z

Deshpande A, Riehle D (2008) The total growth of open source. In: Open Source Development, Communities and Quality. Springer, Boston, pp 197–209. https://doi.org/10.1007/978-0-387-09684-1_16

Dormann CF, Elith J, Bacher S, Buchmann C, Carl G, Carré G, Marquéz JRG, Gruber B, Lafourcade B, Leitão PJ, Münkemüller T, McClean C, Osborne PE, Reineking B, Schröder B, Skidmore AK, Zurell D, Lautenbach S (2013) Collinearity: a review of methods to deal with it and a simulation study evaluating their performance. Ecography 36(1):27–46. https://doi.org/10.1111/j.1600-0587.2012.07348.x

El Emam K, Schneidewind NF (2000) Methodology for Validating Software Product Metrics. Technical Report NCR/ERC-1076, National Research Council of Canada, Ottawa, Ontario, Canada, http://nick.adjective.com/work/ElEmam2000.pdf

Ferreira KAM, Bigonha MAS, Bigonha RS, Mendes LFO, Almeida HC (2012) Identifying thresholds for object-oriented software metrics. J Syst Softw 85(2):244–257. https://doi.org/10.1016/j.jss.2011.05.044. http://www.sciencedirect.com/science/article/pii/S0164121211001385

Field A (2009) Discovering Statistics Using SPSS. SAGE Publications, google-Books-ID: a6FLF1YOqtsC

Gil Y, Lalouche G (2017) On the correlation between size and metric validity. Empir Softw Eng 22(5):2585–2611. https://doi.org/10.1007/s10664-017-9513-5

Hair J (2006) Multivariate Data Analysis. Pearson international edition, Pearson Prentice Hall. https://books.google.se/books?id=WESxQgAACAAJ

Henry S, Selig C (1990) Predicting source-code complexity at the design stage. IEEE Softw 7(2):36–44. https://doi.org/10.1109/52.50772

Henry S, Kafura D, Harris K (1981) On the relationships among three software metrics. In: Proceedings of the 1981 ACM Workshop/Symposium on Measurement and Evaluation of Software Quality. ACM, New York, pp 81–88. https://doi.org/10.1145/800003.807911

Janes A, Lenarduzzi V, Stan AC (2017) A continuous software quality monitoring approach for small and medium enterprises. In: Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion, ACM, New York, ICPE '17 Companion, pp 97–100. https://doi.org/10.1145/3053600.3053618

Jantunen S, Lehtola L, Gause DC, Dumdum UR, Barnes RJ (2011) The challenge of release planning. In: 2011 Fifth International Workshop on Software Product Management (IWSPM), pp 36–45. https://doi.org/10.1109/IWSPM.2011.6046202

Jay G, Hale JE, Smith RK, Hale D, Kraft NA, Ward C (2009) Cyclomatic complexity and lines of code: empirical evidence of a stable linear relationship. J Softw Eng Appl 02(03):137. https://doi.org/10.4236/jsea.2009.23020

Kazman R, Cai Y, Mo R, Feng Q, Xiao L, Haziyev S, Fedak V, Shapochka A (2015) A case study in locating the architectural roots of technical Debt. In: Proceedings of the 37th International Conference on Software Engineering, vol 2. IEEE Press, Piscataway, ICSE '15, pp 179–188. http://dl.acm.org/citation.cfm?id=2819009.2819037

Kendall MG, Gibbons JD (1990) Rank Correlation Methods, 5th edn. Oxford University Press, London

Kim HY (2013) Statistical notes for clinical researchers: assessing normal distribution (2) using skewness and kurtosis. Restorative Dentistry & Endodontics 38(1):52–54. https://doi.org/10.5395/rde.2013.38.1.52. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3591587/

Landman D, Serebrenik A, Bouwers E, Vinju JJ (2016) Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions. Journal of Software: Evolution and Process 28(7):589–618. https://doi.org/10.1002/smr.1760. http://onlinelibrary.wiley.com/doi/10.1002/smr.1760/abstract

Lethbridge TC, Sim SE, Singer J (2005) Studying software engineers: data collection techniques for software field studies. Empir Softw Eng 10(3):311–341. https://doi.org/10.1007/s10664-005-1290-x

Letouzey J, Ilkiewicz M (2012) Managing technical debt with the sqale method. IEEE Softw 29(6):44–51. https://doi.org/10.1109/MS.2012.129

Lincke R, Lundberg J, Löwe W (2008) Comparing software metrics tools. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis, ACM, New York, ISSTA '08, pp 131–142. https://doi.org/10.1145/1390630.1390648

Louridas P, Spinellis D, Vlachos V (2008) Power laws in software. ACM Trans Softw Eng Methodol 18(1):2:1–2:26. https://doi.org/10.1145/1391984.1391986

Mamun MAA, Berger C, Hansson J (2017) Correlations of software code metrics: an empirical study. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, ACM, New York, IWSM Mensura '17, pp 255–266. https://doi.org/10.1145/3143434.3143445

McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng SE-2(4):308–320. https://doi.org/10.1109/TSE.1976.233837

Meloun M, Militký J, Hill M, Brereton RG (2002) Crucial problems in regression modelling and their solutions. Analyst 127(4):433–450. https://doi.org/10.1039/B110779H. https://pubs.rsc.org/en/content/articlelanding/2002/an/b110779h

Meneely A, Smith B, Williams L (2013) Validating software metrics: a spectrum of philosophies. ACM Trans Softw Eng Methodol 21:4:24,1–24:28. https://doi.org/10.1145/2377656.2377661

Meulen MJPvd, Revilla MA (2007) Correlations between internal software metrics and software dependability in a large population of small C/C++ programs. In: The 18th IEEE International Symposium on Software Reliability (ISSRE '07), pp 203–208. https://doi.org/10.1109/ISSRE.2007.12

Park HM (2009) Univariate analysis and normalitytest using sas, stata, and spss Technical Working Paper The University Information Technology Services (UITS) Center for Statistical and Mathematical Computing, Indiana University. https://scholarworks.iu.edu/dspace/handle/2022/19742

Riaz M, Mendes E, Tempero E (2009) A systematic review of software maintainability prediction and metrics. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society, Washington, ESEM '09, pp 367–377. https://doi.org/10.1109/ESEM.2009.5314233

Runeson P, Höst M (2008) Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng 14(2):131–164. https://doi.org/10.1007/s10664-008-9102-8

Saini S, Sharma S, Singh R (2015) Better utilization of correlation between metrics using Principal Component Analysis (PCA). In: 2015 Annual IEEE India Conference (INDICON), pp 1–6. https://doi.org/10.1109/INDICON.2015.7443299

Schroeder J, Berger C, Staron M, Herpel T, Knauss A (2016) Unveiling anomalies and their impact on software quality in model-based automotive software revisions with software metrics and domain experts. In: Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016. ACM, New York, pp 154–164. https://doi.org/10.1145/2931037.2931060

Shevlyakov GL, Vilchevski NO (2002) Robustness in Data Analysis: criteria and methods, Modern Probability and Statistics. VSP BV

Shihab E, Jiang ZM, Ibrahim WM, Adams B, Hassan AE (2010) Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. In: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ACM, p 4

Succi G, Pedrycz W, Djokic S, Zuliani P, Russo B (2005) An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. Empir Softw Eng 10(1):81–104. https://doi.org/10.1023/B:EMSE.0000048324.12188.a2

Tashtoush Y, Al-Maolegi M, Arkok B (2014) The Correlation among Software Complexity Metrics with Case Study. arXiv:1408.4523

Taylor R (1990) Interpretation of the correlation coefficient: a basic review. Journal of Diagnostic Medical Sonography 6(1):35–39

Wheeldon R, Counsell S (2003) Power law distributions in class relationships. In: Proceedings Third IEEE International Workshop on Source Code Analysis and Manipulation, pp 45–54. https://doi.org/10.1109/SCAM.2003.1238030

Xu W, Hou Y, Hung Y, Zou Y (2013) A comparative analysis of spearman's rho and kendall's tau in normal and contaminated normal models. Signal Process 93(1):261–276. https://doi.org/10.1016/j.sigpro.2012.08.005. http://www.sciencedirect.com/science/article/pii/S0165168412002721

Zhou Y, Leung H, Xu B (2009) Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Trans Softw Eng 35(5):607–623. https://doi.org/10.1109/TSE.2009.32

Md Abdullah Al Mamun is a Ph.D. candidate at the Software Engineering Division of Chalmers University of Technology, Sweden. He received a Licentiate in Software Engineering from CHALMERS and a M.Sc. in Software Engineering from Blekinge Institute of Technology, Sweden. His research interests include software metrics and technical debt.

Christian Berger received his Ph.D. degree from RWTH Aachen University, Germany, in 2010. He coordinated the research project for the vehicle "Caroline", which participated in the 2007 DARPA Urban Challenge Final. He also co-led Chalmers Truck Team during the 2016 Grand Cooperative Driving Challenge (GCDC), and is one of the two leading architects behind OpenDLV. He is Associate Professor at the Department of Computer Science and Engineering, University of Gothenburg, Sweden. His research expertise is on distributed realtime software, microservices for embedded systems and cyber-physical systems, and continuous integration/deployment/experimentation for embedded systems.

Jörgen Hansson is a professor of computer science at University of Skövde, Sweden, where is he is the head of the School of Informatics. Prior to joining Skövde, he was a professor in software engineering, founder and co-director of the Software Research Center at Chalmers University between 2010-2013. Between 2005-2010 he was with the Software Engineering Institute, Carnegie Mellon University, USA, leading the performance-critical systems initiative and the architecture-centric engineering initiative. Between 2000-2005 he was with the Linköping University, where he was the director of the national graduate school in computer science. His research interests are in software engineering and data/resource management focusing on the embedded real-time systems domain; he has published 100+ papers in these areas. He received his Ph.D. from Linköping University in 1999, and a M.Sc. and B.Sc. from University of Skövde in 1992 and 1993, respectively.

Affiliations

Md Abdullah Al Mamun[1] · Christian Berger[1] · Jörgen Hansson[2]

Christian Berger
christian.berger@chalmers.se

Jörgen Hansson
jorgen.hansson@his.se

[1] Department of Computer Science and Engineering, Chalmers | University of Gothenburg, Gothenburg, Sweden

[2] School of Informatics, University of Skövde, Skövde, Sweden